AZZURRA ZAPPELLI

PROTOTIPO DI UN SERVIZIO PER LA RACCOLTA E LA VISUALIZZAZIONE DI OSSERVAZIONI AMBIENTALI NELL'AMBITO DELLA GLACIOLOGIA

PROTOTIPO DI UN SERVIZIO PER LA RACCOLTA E LA VISUALIZZAZIONE DI OSSERVAZIONI AMBIENTALI NELL'AMBITO DELLA GLACIOLOGIA

AZZURRA ZAPPELLI

RELATORI:
Prof. Avvenuti Marco
Ing. Vecchio Alessio
Ing. Cesarini Daniel

Università di Pisa
Ingegneria Informatica
Corso di laurea specialistica in Ingegneria Informatica

A.A.
2010/2011

L'amore uccide ciò che siamo stati perchè si possa diventare ciò che non eravamo.

— Sant'Agostino

A Te che con la tua assenza mi hai insegnato e dato più di chiunque altro.

1978 – 1996

ABSTRACT

This work is about a service for remote retrieval of environmental data. The data comes from an Automatic Weather Station (AWS) placed on a glacier. Glaciology, the science that studies glaciers, already uses electronic measurement devices. This work helps out scientists by providing a simple way to get the measurement data in real-time. While working on this thesis an application was developed that acts as a bridge between the measurement devices and a standard framework, the Sensor Observation Service (SOS). SOS standardizes storage and retrieval of environmental observations.
The work of glaciologists is eased thanks to a WebSite that allows to display raw data and its graphical plots.

RIASSUNTO

La tesi tratta la progettazione di un servizio per il recupero da remoto di dati ambientali da una Automatic Weather Station (AWS) situata su di un ghiacciaio. L'idea è di facilitare gli studi dei geologi mediante strumenti più semplici che forniscano dati in tempo reale. Nella glaciologia, la scienza che studia i ghiacciai, sono già usati apparecchi di misura elettronici. Nella tesi è stata sviluppata un'applicazione che si comporta da intermediario tra tali strumenti e il framework Sensor Observation Service (SOS). Il SOS standardizza il recupero e l'immagazzinamento di osservazioni ambientali.
Per semplificare il lavoro dei glaciologi è stato sviluppato un sito web attraverso cui accedere ai dati e vedere i grafici relativi agli ultimi giorni di misura.

INDICE

ELENCO DELLE FIGURE

ELENCO DELLE TABELLE

LISTINGS

ACRONIMI

API	Application Programmable Interface
ARGOS	Advanced Research and Global Observation Satellite
AT	ATtention

AWS	Automatic Weather Station
BASIC	Beginner's All purpose Symbolic Instruction Code
DBMS	DataBase Management System
DOM	Document Object Model
E-R	Entity-Relationship
EDCS	Environmental Data Coding Specification
ESA	European Space Agency
FOI	Feature Of Interest
GEO	Geostationary Earth Orbit
GLOF	Glacial Lake Outburst Flood
GOES	Geostationary Operational Environmental Satellite
GNSS	Global Navigation Satellite System
GSM	Global System for Mobile communications
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
INMARSAT	INternational MARitime Satellite organization
JAR	Java ARchive
JDBC	Java DataBase Connectivity
LEO	Low Earth Orbit
LOS	Line Of Sight
MEO	Medium Earth Orbit
MySQL	My Structured Query Language
OGC	Open Geospatial Consortium
O&M	Observations & Measurements Schema
PHP	PHP:Hypertext Preprocessor
PSTN	Public Switched Telephone Network
RN	Record Number
RS232	Recommended Standard 232

SAS	Sensor Alert Service
SensorML	Sensor Model Language
SEDRIS	Synthetic Environment Data Representation and Interchange Specification
SMTP	Simple Mail Transfer Protocol
SOS	Sensor Observation Service
SPS	Sensor Planning Service
SQL	Structured Query Language
SWE	Sensor Web Enablement
TML	Transducer Markup Language
TS	Time Stamp
UML	Unified Modelling Language
URN	Uniform Resource Name
WGMS	World Glacier Monitoring Service
WNS	Web Notification Service
WSN	Wireless Sensor Network
XML	eXstensible Markup Language

Parte I

VISIONE AD ALTO LIVELLO

Introduzione alla tesi partendo dalle motivazioni, analizzando lo stato dell'arte e arrivando alla progettazione del servizio.

1 INTRODUZIONE

La tecnologia si presenta sempre di più come la risposta alle necessità dell'uomo, entrando a far parte della quotidianità in modo pervasivo e mobile. Sono molteplici i campi dove la tecnologia sta dando un forte contributo e, negli ultimi anni, un settore dove questa collaborazione si è fatta più proficua è quello ambientale.
L'evoluzione dei sensori e l'integrazione degli stessi in sistemi di comunicazione, hanno reso possibile l'osservazione di fenomeni naturali con una granularità spazio/temporale fino ad ora impensabile, grazie all'installazione di Wireless Sensor Network (WSN) o datalogger nei luoghi di interesse.
Le WSN sono reti di nodi sensori che comunicano senza l'uso di infrastrutture cablate. I sensori si occupano dell'acquisizione dei dati nell'ambiente in cui sono immersi. I dati raccolti vengono poi resi disponibili agli utenti remoti in tempo quasi reale, ad esempio, con cadenza giornaliera. È possibile rendere i sensori ri-programmabili da remoto; ovvero, permettere ad un utente remoto di cambiare alcuni parametri come la frequenza di campionamento, iniettando dei comandi nella rete.
I datalogger sono acquisitori di dati non dotati della possibilità di comunicare:
i dati raccolti vengono immagazzinati su supporti interni di memoria e scaricati dall'utente che deve recarsi in loco allo stesso.
Ad oggi sono molte le installazioni di datalogger, soprattutto in ambienti ostici, limitando di fatto l'intervento umano solo all'installazione dell'apparecchio e allo scaricamento dei dati.
L'idea alla base di questa tesi è quella di far evolvere una stazione meteorologica, attualmente funzionante come un semplice datalogger, in un agglomerato di nodi sensori che pubblicano le loro osservazioni[1] su un sito web appoggiandosi ad un servizio standard quale il *Sensor Observation Service (SOS)*.
La relativa novità della tecnologia dei sensori e l'ecletticità del suo uso, ne ha reso difficile il processo di standardizzazione e omogenizzazione. Vista l'importanza dell'uso di architetture standard per la durabilità di un progetto e per la sua integrazione in contesti diversi apportando il minor numero di modifiche possibili, si fa riferimento all'iniziativa *Sensor Web Enablement (SWE)*, che porta avanti un processo di standardizzazione avviato dall'**Open Geospatial Consortium (OGC)** [6].

1 Il dato prodotto al termine del processo di misurazione eseguito da un sensore e del successivo processo di codifica di tale dato come *n-upla* di valori numerici, con n dipendente dalla tipologia di fenomeno misurato

1.1 MOTIVAZIONI

In ambienti ostili, quali quello della glaciologia, l'intervento tecnologico si presenta come uno strumento indispensabile. Questo è apparso evidente in seguito ad un incontro con dei geologi dell'università di Padova e di Pisa che lavorano con due stazioni meteorologiche, operative da qualche anno, su due ghiacciai italiani:

- Ghiacciaio de La Mare [Trentino Alto Adige]
- Ghiacciaio dell'Ortles [Trentino Alto Adige]

Entrambe le stazioni sono composte da un datalogger con annessi diversi sensori per il monitoraggio ambientale.
Dall'incontro sono emersi i limiti di queste installazioni. Come detto nell'introduzione, per recuperare i dati dai datalogger, è necessario recarsi sul posto e scaricarli dalla scheda di memoria in cui sono immagazzinati. Questo, oltre a richiedere spese e tempo per il viaggio, è anche impossibile nel periodo che va da ottobre a maggio, in quanto le stazioni meteorologiche sono irraggiungibile. I geologi si trovano costretti ad aspettare senza avere la garanzia che tutto stia funzionando correttamente.
Appaiono ora evidenti le motivazioni di questa tesi:
dotando le stazioni meteorologiche della capacità di comunicare i viaggi al sito sono limitati a quelli di manutenzione, i dati sono reperibili con cadenza giornaliera da un qualsiasi apparecchio dotato di connessione a Internet ed è possibile monitorare lo stato del sistema.

1.2 STATO DELL'ARTE

Prima di affrontare la progettazione del servizio, è stato eseguito un allineamento su quello che è ad oggi l'uso della tecnologia nell'ambito del monitoraggio ambientale con particolare riferimento alla glaciologia.

1.2.1 *Tecnologia & Glaciologia*

Molti sono i progetti attivi nell'ambito del monitoraggio ambientale. Tra gli strumenti più usati:

- Datalogger
- Satelliti
- WSN
- Field Server
- Probe

DATALOGGER Sono i primi apparecchi che sono stati usati nell'ambito del monitoraggio ambientale. Molte installazioni ancora oggi ne fanno uso, ad esempio il progetto *PermaNET* sulle Alpi [12].

Figura 1: Recupero dati da un datalogger nel progetto PermaNET.

SATELLITI Altra tecnologia ampiamente utilizzata si basa sull'uso dei satelliti. Questa è definita *Remote Sensing*, in quanto l'acquisizione dell'osservazione è effettuata senza contatto fisico con il fenomeno misurato.
I principali progetti sono avviati dall'*European Space Agency (ESA)* [3] e dal *World Glacier Monitoring Service (WGMS)* [8].

WSN Durante la documentazione sullo stato dell'arte sono emersi molti progetti che usano le WSN. Questa sembra essere la tecnologia su cui si investe di più in questo ambito, grazie ai suoi innumerevoli punti di forza, tra cui:

- Elevata densità spaziale e temporale nel sensing dell'ambiente in cui tali reti sono installate
- Accesso quasi real-time ai dati acquisiti
- Ri-programmabilità da remoto
- Adattabilità ad ambienti ostili in quanto non richiedono l'intervento umano dopo la loro installazione

- Economicità del nodo sensore

La relativa novità di questa tecnologia lascia ancora spazio a limiti e quindi a sfide:

- Tempo di vita della rete vista l'alimentazione a batteria
- Sincronizzazione temporale tra i vari nodi
- Comunicazione tra i sensori in ambienti particolarmente ostili

Tra i progetti più interessanti *PermaSense* [19], svolto sulle Alpi e *Sierranet* [25], svolto sulle catene montuose della Sierra Nevada.

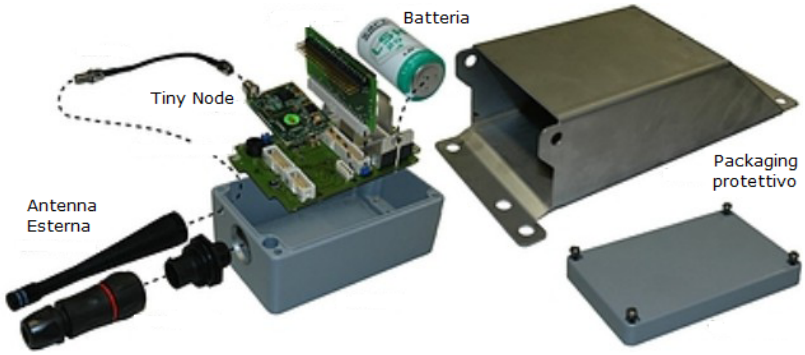

Figura 2: Nodo sensore usato nel progetto PermaSense.

FIELD SERVER Si tratta di particolari stazioni meteorologiche a cui è possibile annettere diversi tipi di sensori. Sono dotati di pannelli solari che assicurano loro l'indipendenza dalla rete elettrica. Comunicano via *wi-fi*.
Sono usati sulla catena montuosa dell'Everest per monitorare il lago *Imja* [23] in modo da prevenire, o quanto meno limitare, i danni in caso di Glacial Lake Outburst Flood (GLOF) [2].

Figura 3: Field Server usato nel progetto di monitoraggio del lago Imja.

2 Fenomeno rappresentato dall'esondazione di un lago in seguito allo scioglimento di ghiacciai limitrofi

PROBE Sono particolari nodi sensori adibiti allo studio delle dinamiche del movimento dei ghiacciai e del *till* [3].
I probe sono usati nel progetto *GlacsWeb* [20] [4] e rappresentano una tecnologia molto innovativa ed unica nel suo genere. Sono inseriti all'interno dei ghiacciai, quindi ne seguono i movimenti, trasmettendo i dati alla stazione base in superficie come si evince dalla Figura 4.

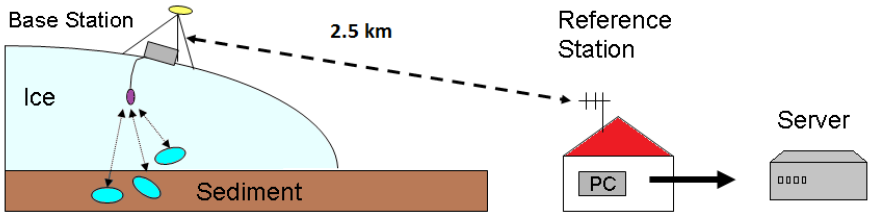

Figura 4: Architettura ad alto livello del progetto *GlacsWeb*.

1.2.2 *Standard*

Per quanto riguarda gli standard, sia per la descrizione dei sensori che per i dati che acquisiscono, non è ancora stata raggiunta una piena maturità. L'eterogeneità di questa tecnologia, ha spinto l'OGC a stendere delle specifiche che riguardano l'uso dei sensori nell'ambito geospaziale.
Nella tesi viene fatto riferimento ai framework specificati nell'iniziativa *SOS* dell'OGC, visto l'importanza di creare servizi standard e l'autorità riconosciuta all'OGC.

3 Indica rocce deposte da un ghiacciaio in ambiente glaciale o periglaciale o in regioni polari.

2 SITO E STRUMENTAZIONE

Questo capitolo illustra i due siti presi in esame e gli strumenti usati nelle installazioni. I siti si trovano in Trentino Alto Adige e sono composti da una Automatic Weather Station (AWS), ovvero una stazione meteorologica che non richiede l'intervento umano per raccogliere i dati ambientali.

2.1 SITO VAL DE LA MARE

La AWS è situata sul ghiacciaio de La Mare sul fronte Alpino presente nel Trentino Alto Adige ad un'altitudine di 2970 metri. L'installazione è avvenuta il sedici luglio del 2007.
Il sito è raggiungibile in circa tre ore da Malga Mare e, passato il rifugio Larcher, è necessario procedere a piedi. É necessario affrontare un dislivello di circa 1000 metri, quindi è richiesto un po' di allenamento. L'attrezzatura necessaria comprende scarponi da montagna, ramponi, piccozza e un abbigliamento adeguato. I geologi che scaricano i dati dalla stazione, si recano in loco circa sette volte l'anno, in un arco temporale che va da fine maggio a inizio ottobre. Durante queste visite si occupano anche della manutenzione della AWS e dei sensori annessi. Ad esempio, una volta l'anno, i sali essicanti nei sensori a radiazione devono essere sostituiti.
Per completezza sono monitorate anche le portate dei torrenti alimentati dal ghiacciaio de La Mare grazie a due piezometri:
uno nei pressi del rifugio Larcher e l'altro al pian Venezia.
Da sottolineare la mancanza di copertura delle reti cellulari.

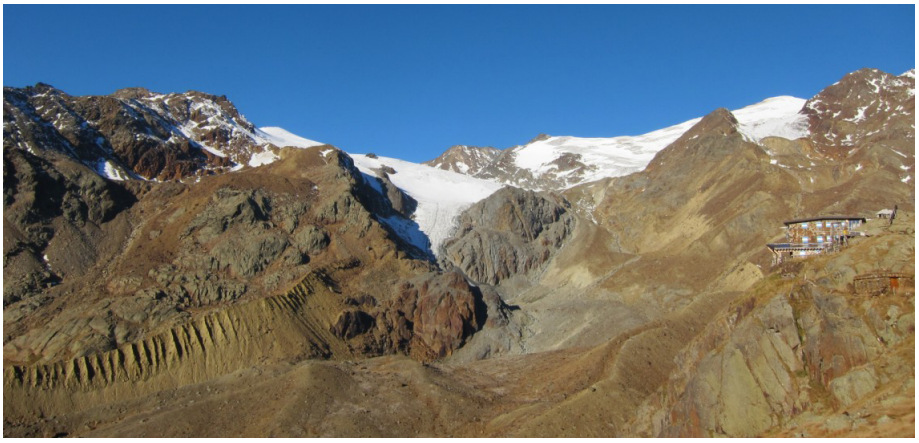

Figura 5: Il ghiacciaio de La Mare con a destra il rifugio Larcher.

2.2 SITO ORTLES

La AWS è situata sul ghiaccio dell'Ortles ad un'altitudine di 3840 metri. L'Ortles è una cima del fronte Alpino del Trentino Alto Adige sul confine con la Svizzera. L'installazione è recente e risale al primo ottobre del 2011.
La raggiungibilità di questo sito è molto più critica rispetto a quello in Val de La Mare. Infatti, l'unica via di accesso via terra è un percorso alpinistico che richiede esperienza e rende difficoltoso il trasporto degli strumenti. Solitamente, viene noleggiato un elicottero benché economicamente dispendioso. Per questi motivi le spedizioni sul sito sono ridotte, circa due l'anno, e affrontate nei mesi estivi.
É presente copertura cellulare, anche se molto debole.

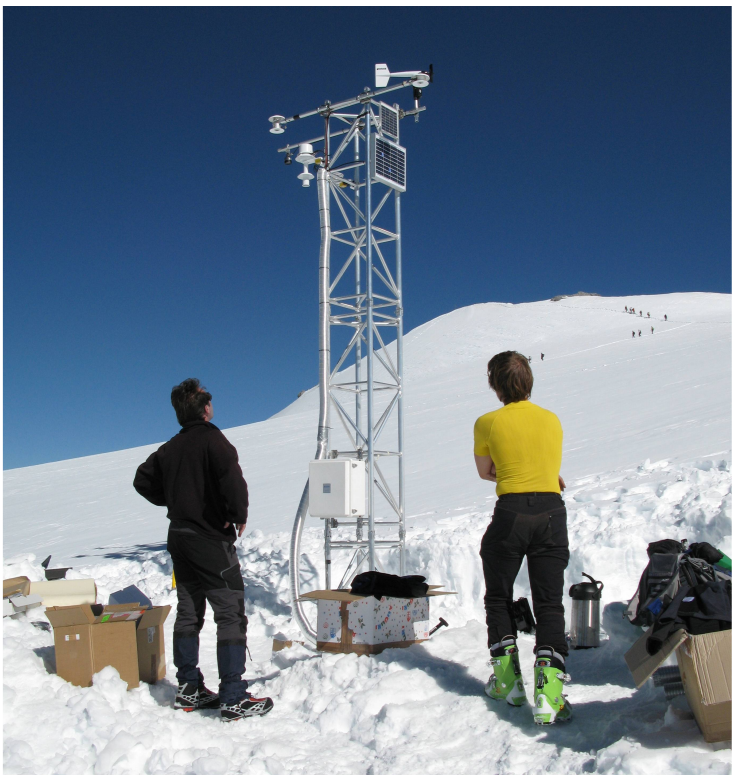

Figura 6: Stazione meteorologica dell'Ortles.

2.3 CR1000

Il *CR1000* [16] è un acquisitore dati a cui è possibile annettere molteplici sensori. É prodotto dalla **Campbell Scientific** [1], azienda con sedi sparse in tutto il mondo che si occupa di produrre tecnologia per il monitoraggio ambientale; soprattutto sensori, datalogger e software annessi.
Un overview del sistema è dato dal seguente schema:

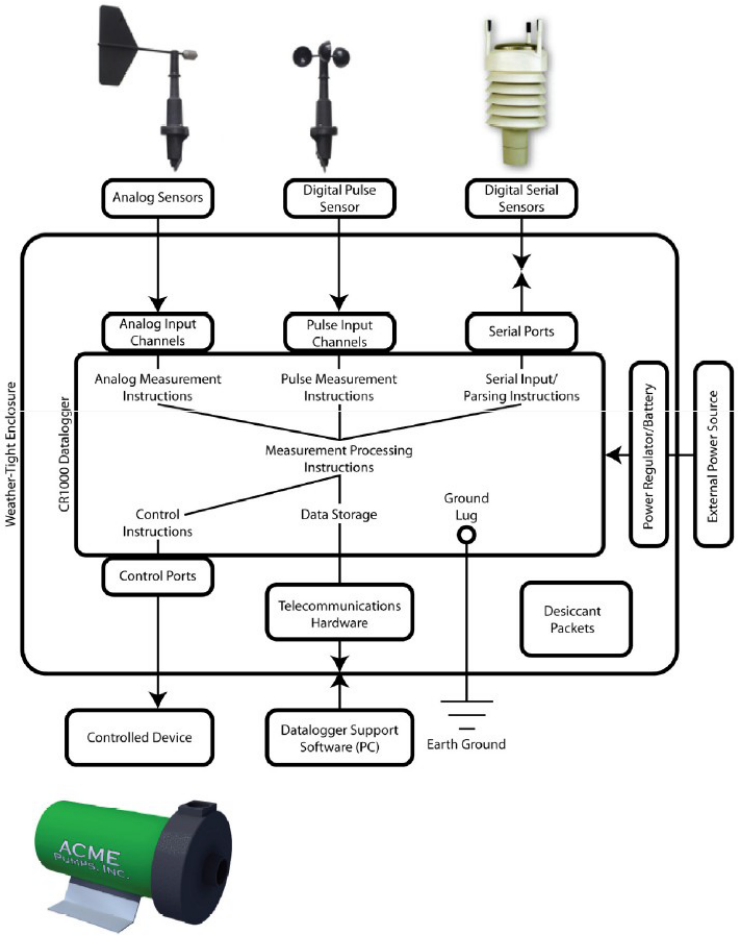

Figura 7: Overview del CR1000.

Lo schema mostra come sia possibile collegare al *CR1000* sensori, sistemi di alimentazione [Batterie Ricaricabili, Pannelli Solari] e hardware per la telecomunicazione. A tale scopo è dotato di un *Wiring Panel*:

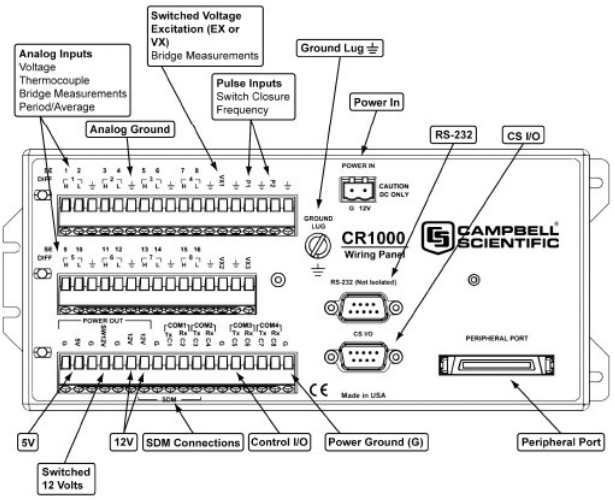

Figura 8: Wiring Panel del CR1000.

2.3.1 *Suite Software*

Sono disponibili diverse suite di programmi proprietarie della **Campbell Scientific**:

- **PC200W** - È *freeware* e comprende i seguenti tool:
  - **Short Cut**, è un editor grafico per la creazione di semplici programmi CRBASIC.
  - **Device Configuration Utility**, contiene un elenco dei dispositivi che riconosce con una descrizione e ne permette la configurazione.
- **PC400** - É proprietario e comprende:
  - Un editor CRBASIC per scrivere programmi più complessi.
  - Supporto ad una varietá di opzioni di telecomunicazione e collezione di dati.
  - **Transformer Utility** è un'utilitá per convertire in programmi *CRBASIC* programmi *Edlog* [1].
- **LoggerNet** - É proprietario e comprende funzionalitá più complesse per quanto riguarda il supporto a:
  - Opzioni di telecomunicazione [ad esempio: phone-to-RF]
  - Personalizzazione nel monitoraggio / visualizzazione dei dati
  - Salvataggio dei dati in formati [incluso eXstensible Markup Language (XML)] che possono essere importati da *third-party package di analisi*

Ci sono anche altri package:

- **LoggerNet Admin**, indicato per chi deve gestire grandi reti di datalogger. Aggiunge funzionalitá tra le quali sicurezza di rete, gestione della rete da un computer remoto, possibilitá di lanciare più istanze di uno stesso cliente.
- **LoggerNet Remote**, è la suite completa di applicazioni LoggerNet Admin Client che consente di gestire, da un computer remoto, una rete di datalogger esistente.
- **LoggerNet per Linux**, fornisce una soluzione per chi desidera usare il server LoggerNet in ambiente Linux. Il pacchetto include una versione linux del Server LoggerNet e una copia del LoggerNet remoto.

1 Vecchio linguaggio di programmazione dei datalogger della **Campbell Scientific**. È sempre supportato ma è meno intuitivo sia nella scrittura che nella comprensione

Nel *CR1000* possono essere presenti in memoria più programmi ma solo uno alla volta può essere eseguito.
Esiste un limite alla dimensione dei programmi utente pari a 490 kB.

2.4 SENSORI

I sensori in dotazioni riguardano il monitoraggio ambientale.
Di seguito sono illustrati quelli presenti nella AWS nel sito in Val de La Mare.

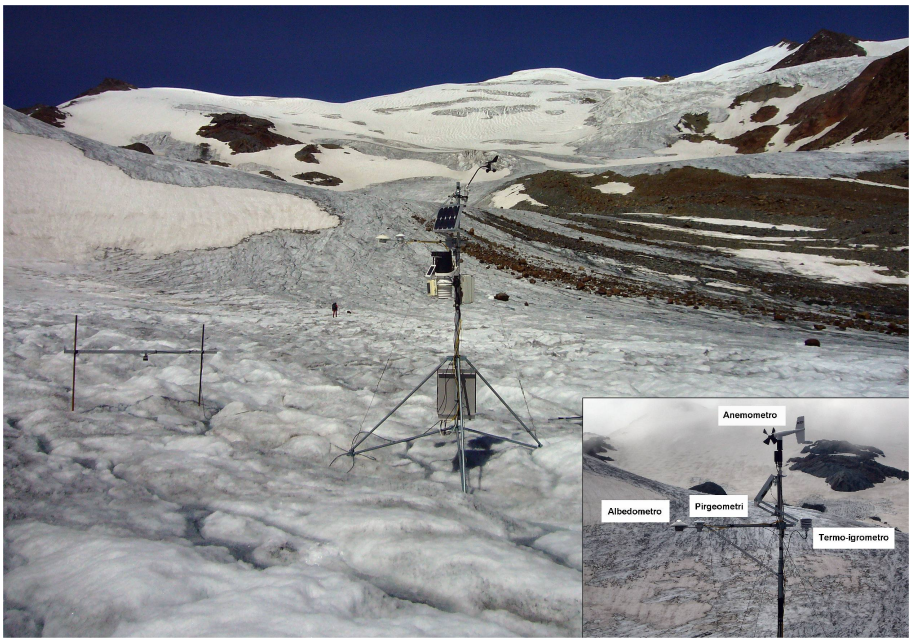

Figura 9: AWS sul ghiacciaio de La Mare.

2.4.1 *Albedometro*

L'*albedometro LP Pyra05* é prodotto dalla **Delta Ohm**. Questo strumento misura la radiazione totale netta e l'albedo[2] dei terreni. Per farlo si serve di due piranometri[3] accoppiati in maniera d'avere la stessa sensibilità:

- Uno rivolto verso l'alto che misura la luce riflessa
- Uno rivolto verso il basso che misura la luce incidente sul terreno

Dal cavo di uscita si ottengono i segnali dei due piranometri che possono essere elaborati per ottenere la grandezza fisica di interesse.

2 rapporto tra la radiazione diffusa da una determinata superficie e la quantità di radiazione che arriva sulla superficie

3 Misurano la radiazione globale, ovvero quella data dalla radiazione *diffusa* più quella *diretta*

Da tenere presente che l'accuratezza delle misure dipende dalle pulizia esterna delle cupole dei piranometri.

Figura 10: Albedometro LP Pyra05.

2.4.2 *Anemometro*

L'*anemometro 05103* é prodotto dalla **R.M. Young**. Questo strumento misura la velocità e la direzione del vento.
Il sensore per la misura della velocità é formato da quattro eliche elicoidali che, ruotando, producono un segnale sinusoidale con una frequenza direttamente proporzionale alla velocità del vento.
La misura può essere dedotta anche contando gli impulsi:

- Tre impulsi corrispondono ad un giro dell'elica

La direzione é misurata tramite una banderuola.
Da tenere presente che è soggetto a guasti; in particolare:

- Il potenziometro
- I cuscinetti a sfera di precisione

Figura 11: Anemometro RM Young 05103.

2.4.3 *Nivometro*

Il *nivometro ad ultra suoni SR50A* é prodotto dalla **Campbell Scientific**. Questo strumento misura, approssimativamente, le precipitazioni nevose.
È un sensore acustico che misura il 'tempo di volo' per determinare la profondità della neve o dell'acqua.
Per misurare il 'tempo di volo' calcola il tempo trascorso tra l'emissione di un impulso ad ultrasuoni e il suo ritorno. [*É necessario conoscere la temperatura dell'aria per correggere la misura*]

Figura 12: Nivometro ad ultra suoni SR50A.

2.4.4 *Pirgeometro*

Il *pirgeometro CGR3* é prodotto dalla **Kipp&Zonen**. Questo strumento misura l'intensità della radiazione emessa per irraggiamento termico dal suolo e dall'aria.
Collegati al *CR1000* ce ne sono due.
Da tenere in considerazione che è raccomandata una ricalibrazione ogni due anni.

Figura 13: Pirgeometro CGR3.

2.4.5 *Termistore*

Il *sensore di temperatura T-107* é prodotto dalla **Campbell Scientific**. É un termistore in grado di misurare la temperatura di diversi mezzi tra cui aria, suolo e acqua. In questo contesto é usato per l'aria.
Collegati al *CR1000* ce ne sono due. Uno é alloggiato nello schermo ventilato[4] passivo mentre l'altro in quello attivo.

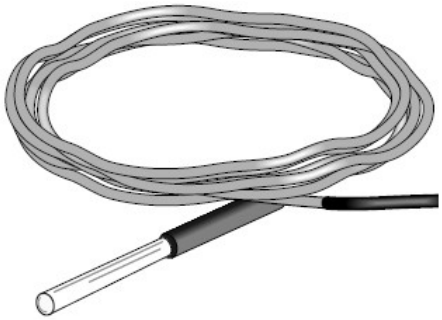

Figura 14: Termistore T-107.

2.4.6 *Termoigrometro HMP45C*

Il *Termoigrometro HMP45C* é prodotto dalla **Vaisala**. È uno strumento che permette di misurare due grandezze inerenti l'aria:

- La temperatura [*termo*]
- L'umidità relativa [*igrometro*]

È composto da due sensori:

- Sensore di temperatura [*PRT*] che sfrutta la relazione conosciuta tra la resistenza elettrica di alcuni materiali, in questo caso il Platino, con la variazione della temperatura per dedurne il valore.
- Sensore capacitativo [*Humicap H-chip*] costituito da un polimero, sensibile all'umidità, su un supporto di vetro che giace tra due strati metallici. In altre parole, si forma un condensatore tra le due armature metalliche e il polimero funge da dielettrico. Il polimero, assorbendo acqua, cambia la costante dielettrica e, di conseguenza, la capacità del condensatore in modo proporzionale all'umidità.

Il *termoigrometro HMP45C* é alloggiato in uno schermo ventilato attivo ed è accesso e spento tramite un interruttore controllato via software. Da tenere presente che è raccomandata una calibrazione annuale.

4 Mantiene la corretta temperatura dell'aria e protegge i sensori da pioggia e neve. L'aggettivo *attivo* si riferisce alla presenza di una ventola che si attiva automaticamente quando il cielo é soleggiato, altrimenti si dice *passivo*

Figura 15: Termoigrometro HMP45C.

2.4.7 *Termoigrometro CS215*

Il *Termoigrometro CS215* é prodotto dalla **Campbell Scientific**. Il principio di funzionamento è analogo a quello del *termoigrometro HMP45C* [vedi sottosezione 2.4.6]. Il *termoigrometro CS215* é alloggiato in uno schermo ventilato passivo. Viene usato per confrontare le misure prese dal *termoigrometro HMP45C*.

Figura 16: Termoigrometro CS215.

2.5 SISTEMA DI ALIMENTAZIONE

Il sistema di alimentazione assicura alla stazione meteorologica l'energia necessaria per sopravvivere senza l'attacco alla rete di alimentazione.

2.5.1 *Batteria Ricaricabile*

La batteria ricaricabile adoperata é la *BP24E-LA* distribuita dalla **Yuasa** ed insieme c'é un regolatore di carica.
Le sue caratteristiche salienti sono:

- Tensione -> 12 V
- Capacità -> 24 Ah
- Dimensioni -> 175 x 166 x 125 mm
- Peso -> 10.2 Kg

Figura 17: Batteria BP24E-LA.

Per quanto riguarda il regolatore di carica é interessante osservare la relazione che lega la capacità di carica alla temperatura in cui si trova ad operare:

Tabella 1: Tabella sulla dipendenza tra Capacità e Temperatura.

Temperatura (°C)	**% a 20 °C di Capacità**	**Capacità (Ah)**
40	110	26.4
20	100	24.0
0	84	20.2
−20	72	17.3

Al diminuire della temperatura la capacità decresce notevolmente. Questo non é un fattore positivo visto che il periodo in cui il sito rimane irraggiungibile é proprio quello invernale. Da sottolineare che in questi anni di attività delle due stazioni meteorologiche [sezione 2.1 e sezione 2.2] non ci sono mai stati problemi legati all'alimentazione.

2.5.2 *Pannello Solare*

Il pannello solare adoperato é il *SP20* distribuito dalla **Campbell Scientific**.
Le sue caratteristiche salienti sono:

- Potenza -> 20W
- Corrente di picco -> 1170 mA
- Dimensioni -> 420 x 500 mm [*Escluso sostegno*]
- Peso -> 4.8 Kg [*Incluso sostegno*]

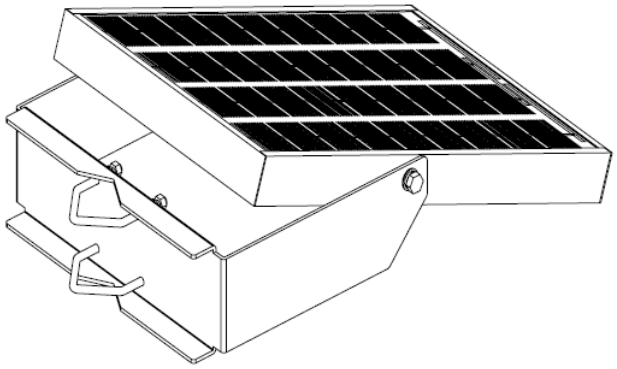

Figura 18: Pannello Solare SP20.

Per quanto riguarda il montaggio, il pannello solare, ha un'inclinazione regolabile tramite il sostegno. Per un'installazione di lunga durata e soggetta a precipitazioni nevose, é consigliata un'inclinazione maggiore di quindici gradi.

2.6 SISTEMA DI COMUNICAZIONE

Dotare le stazioni meteorologiche della capacità di comunicare renderebbe i dati, acquisiti con il sensing dei sensori annessi, accessibili agli utenti interessati. É un aspetto molto importante nella progettazione del sistema in quanto punto critico del consumo di energia e quindi del suo tempo di vita.
É stato fatto uno studio sulle opzioni di comunicazione permesse dall'ostile ubicazione delle due stazioni meteorologiche per scegliere la migliore tipologia di modem.

2.6.1 *GSM*

La prima tipologia di modem selezionata è quella con supporto Global System for Mobile communications (GSM). Il GSM è uno standard della comunicazione su reti cellulari. La copertura offerta dai principali

gestori telefonici è molto ampia e i costi della comunicazione relativamente bassi.

2.6.1.1 *Fastrack Xtend*

Il *Fastrack Xtend* è un modem prodotto dalla **Sierra Wireless Airlink**. É stato selezionato per la comunicazione *GSM* per tre ragioni principali:

- É il modem indicato dalla **Campbell Scientific** [17] come modulo aggiuntivo per il *CR1000*
- I geologi proprietari dei siti avevano già a disposizione questo modem
- Ha un prezzo contenuto, intorno ai 130 €

Figura 19: Modem Fatrack Xtend.

Da sottolineare che supporta i comandi *AT*[5].

2.6.2 *Comunicazione Satellitare*

Le reti satellitari sono reti di telecomunicazione che attraverso collegamenti radio satellitari permettono lo scambio di dati. I componenti che permettono questo tipo di comunicazione sono:

- Il trasmettitore
- Il ricevitore
- Il satellite in orbita
- La stazione di controllo a terra, effettua tutte le operazioni di tracking[6], telemetria e guida del satellite.

5 insieme di comandi per configurare e programmare modem fonici tramite interfaccia seriale

6 Determinazione della posizione e della velocità del satellite

Il trasmettitore e il ricevitore fanno parte del *segmento terrestre* mentre il Satellite e la stazione di controllo del *segmento spaziale*. I satelliti sono classificabili in base all'orbita che percorrono in:

- Satelliti ad orbita geostazionaria [Geostationary Earth Orbit (GEO)]
- Satelliti ad orbita media [Medium Earth Orbit (MEO)]
- Satelliti a bassa orbita [Low Earth Orbit (LEO)]

L'impronta a terra[7] dipende dalla quota a cui si trova il satellite:
al decrescere della quota diminuisce l'impronta a terra.
Per questo motivo sono necessari più satelliti LEO per avere la copertura di una stessa area in confronto ai satelliti GEO.
I satelliti geostazionari seguono un'orbita geostazionaria:
orbita circolare ed equatoriale, situata ad un'altezza tale che il periodo di rivoluzione di un satellite che la percorre coincide con il periodo di rotazione della terra. Quindi, questi satelliti, appaiono immobili ad un osservatore terrestre.
Per ridurre il consumo energetico, le antenne dei modem che comunicano con satelliti GEO, sono monodirezionali. In questo modo è possibile abbassare la potenza del segnale emesso visto che è tutta convogliata in un'unica direzione.
I satelliti LEO, grazie alla loro bassa quota, permettono comunicazioni più veloci e consumi energetici minori rispetto ai GEO. Generalmente, i modem che usano reti satellitari LEO, sono abbinati con antenne omnidirezionali più semplici da installare in quanto non richiedono un particolare puntamento.
Nel monitoraggio ambientale, dove il consumo energetico è un vincolo molto stringente, la direzione seguita è generalmente quella delle reti satellitari LEO. Però, con l'avanzamento della tecnologia, sono stati prodotti anche modem a basso consumo che si interfacciano con i satelliti geostazionari.
In conclusione sono molte le soluzioni offerte dalla tecnologia satellitare. La scelta non è solo tra LEO e GEO, ma anche tra i modelli di modem delle differenti case produttrici.

2.6.2.1 *MiChroSat 2403*

Il *MiChroSat 2403* è un modem satellitare prodotto dalla **Wireless Innovation** che usa reti satellitari LEO *Iridium*.
É stato selezionato in quanto modem indicato dai fornitori del *CR1000*.
Da sottolineare che supporta i comandi ATtention (AT).

7 in inglese *footprint*, indica l'estensione dell'area di copertura del satellite

Figura 20: Modem Satellitare MiChroSat 2403.

2.6.2.2 *Hughes 9502*

Lo *Hughes 9502* è un modem satellitare prodotto dalla **Hughes** che usa il satellite geostazionario *OMNISAT*.
Le caratteristiche salienti dello *Hughes* sono l'alto rate di trasmissione, l'elevata resistenza all'ambiente glaciale e la modalità *always-on* [8].
Lo *Hughes* usa un'antenna monodirezionale di piccole dimensioni per comunicare con il satellite GEO *OMNISAT*.
Supporta i comandi AT ma non è possibile usare la porta seriale Recommended Standard 232 (RS232) per la trasmissione di dati. Essa è adibita solo al sevizio Global Navigation Satellite System (GNSS).

Figura 21: Modem Satellitare Hughes 9502.

8 Modalità di funzionamento a basso consumo del modem, che mantiene la sessione dati attiva e non richiede particolari procedure di *wake-up*

3

TECNOLOGIA&STANDARD DI RIFERIMENTO

Questo capitolo illustra le tecnologie e gli standard che sono stati adoperati nella tesi.
Per quanto concerne la tecnologia, è fatto riferimento a quella più inusuale; ovvero, è trattato il *CRBASIC* mentre non è fatta parola del ben noto *Java*. Di quest'ultimo saranno analizzati spezzoni di codice giudicati interessanti al fine di comprendere il funzionamento dell'applicazione **Proxy**.

3.1 SENSOR WEB ENABLEMENT

L'**OGC** è un consorzio internazionale, formato da 439 membri, che si occupa della creazione di interfacce standard per la gestione delle reti di sensori.
L'iniziativa *SWE* [15] specifica framework che permettono l'integrazione di sensori eterogenei nascondendone le diversitá sia hardware che software. Questo semplifica l'interazione con le applicazioni clienti che vedono i sensori come un'unica rete *service-oriented* interoperabile e scalabile.
Il *SWE* focalizza l'attenzione sui seguenti processi:

- *Discovery* [1] dei sensori e/o delle osservazioni che soddisfano le richieste di un'applicazione cliente.
- Determinazione delle caratteristiche hardware e software di un sensore.
- Accesso ai parametri del sensore che consentono automaticamente l'elaborazione e la localizzazione delle osservazioni.
- Recupero di osservazioni *real-time* e loro codifica secondo uno standard.
- *Tasking* [2] dei sensori per ottenere osservazioni di interesse.

Il *SWE*, nell'ambito di tali processi, stabilisce le codifiche per la descrizione sia dei sensori che delle osservazioni e definisce le interfacce per la realizzazione di servizi web.
Gli standard e le interfacce attualmente definiti e prototipati sono i seguenti: [*Di cui i primi cinque sono i principali*]

1 processo di rilevazione automatica della presenza delle istanze di una certa entità in rete
2 Campagna di misurazioni

- **Observations & Measurements Schema (O&M)**, modelli e schemi XML standard per la codifica delle osservazioni, real-time e non.
- **Sensor Model Language (SensorML)**, modelli e schemi XML standard per la descrizione dei sensori:
  fornisce le informazioni richieste per il processo di *discovery* dei sensori, i loro parametri regolabili, la localizzazione geografica e l'elaborazione delle osservazioni.
- **Transducer Markup Language (TML)**, modello concettuale e schemi XML per la descrizione dei trasduttori e il supporto dello streaming real-time di dati da e verso reti di sensori.
- **Sensor Observation Service (SOS)**, framework per la richiesta, il filtraggio e il recupero delle osservazioni e delle informazioni associate a una rete di sensori. Funziona da intermediario fra un'applicazione cliente e un database di osservazioni o un canale in cui vengono prodotte in modalità real-time.
- **Sensor Planning Service (SPS)**, framework per la richiesta di produzione di osservazioni secondo un preciso set di parametri (processo di *tasking* dei sensori). Funziona da intermediario fra un'applicazione cliente e un ambiente di gestione diretta dei sensori.
- **Sensor Alert Service (SAS)**, framework per la realizzazione di un sistema *publish-subscribe* [3] che notifichi gli eventi di produzione delle osservazioni alle applicazioni clienti in maniera asincrona.
- **Web Notification Service (WNS)**, framework per l'invio asincrono di messaggi da parte dei servizi del SAS e del SPS.

La sinergia degli standard e dei framework citati è funzionale alla realizzazione di reti di sensori che si integrino nella rete Internet. Questa integrazione rende trasparente alle applicazioni clienti l'eterogeneità che diversifica le due tipologie di rete e provvede ad un accesso alle osservazioni e alle caratteristiche dei dispositivi che le producono secondo una filosofia *plug-and-play*.
Il progresso della tecnologia digitale sta rendendo possibile la dotazione di una connessione cablata o meno pressochè a ogni tipo di

3 Modello per la comunicazione asincrona fra diversi processi, oggetti o altri agenti. In questo schema, mittenti e destinatari di messaggi dialogano attraverso un tramite, detto *dispatcher* o *broker*. Il mittente di un messaggio (*publisher*) non conosce l'identità dei destinatari (*subscriber*) ma si limita a pubblicare il proprio messaggio al *dispatcher*. I destinatari si rivolgono a loro volta al *dispatcher* per registrarsi ad un certo tipo di messaggio. Il *dispatcher* quindi inoltra ogni messaggio inviato da un *publisher* a tutti i *subscriber* sottoscritti.

sensore esistente. Tali connessioni supportano l'accesso remoto ai parametri di controllo, ai dati prodotti, alle informazioni di localizzazione e alle caratteristiche hardware e software dei dispositivi. Se al di sopra della connessione girano protocolli Internet, il linguaggio XML può essere usato per pubblicare descrizioni formali delle *capabilities* dei sensori, della loro posizione geografica e delle interfacce da questi offerte per il tasking e la codifica dei dati. Pertanto le applicazioni clienti possono processare ed interpretare i dati XML, abilitando in rete il *discovery* automatico di sensori e la valutazione delle loro caratteristiche basata sulle descrizioni che questi rendono pubbliche. Tali informazioni consentono anche alle applicazioni di geolocalizzare ed elaborare le osservazioni senza richiedere una conoscenza a priori dei dispositivi che le producono.

3.1.1 *Architettura Generale*

All'interno della tesi è stato utilizzato solo il framework *SOS*, per il recupero e l'inserimento di osservazioni, e il linguaggio *SensorML*, per la descrizione dei sensori da registrare al servizio *SOS*.

3.1.2 *Sensor Observation Service*

Il framework *SOS* [22] fornisce un Application Programmable Interface (API) per la memorizzazione e il recupero dei dati e dei metadati associati ai sensori a lui registrati.
Tali sensori possono essere sia statici che dinamici o sia remoti che in-situ [4]. L'interfaccia con cui si accede ai loro metadati è indipendente dalla classificazione del sensore stesso. Infatti, tutte le *capabilities* sono recuperabili mediante una richiesta di *GetCapabilities*. Questo rende il *SOS* molto elastico ed utile, soprattutto nel *discovery* dei sensori.
Il framework si comporta come uno strato di *middleware* tra le applicazioni clienti e i sensori, nascondendone l'eterogeneità hardware e software realizzando un API a comune.
Le operazioni fornite dal API sono raggruppate in quattro profili:

- *Core*, offre le funzionalità basilari per il funzionamento del framework, ovvero per il recupero delle osservazioni. La sua implementazione è obbligatoria.
- *Transactional*, offre le funzionalità per la registrazione dei sensori e per l'inserimento delle osservazioni. La sua implementazione è opzionale.
- *Enhanced*, offre funzionalità aggiuntive. La sua implementazione è opzionale.

4 A contatto fisico con il fenomeno che misura

- *Entire*, racchiude tutte le operazioni dei precedenti profili.

3.1.3 *Sensor Model Language*

L'obiettivo principale del *SensorML* [14] è quello di definire i processi e i componenti di *processing* associati sia con la misura che con le trasformazioni post-misura delle osservazioni.

3.1.3.1 *Perché usare il SensorML?*

I sensori giocano ormai un ruolo fondamentale in diversi campi tra cui il monitoraggio ambientale, la gestione di emergenze, l'*intelligence*. Sono quindi sempre più immersi nella vita quotidiana, basti pensare che anche il più semplice smartphone è dotato di un accelerometro. Purtroppo il processo di *discovery* raramente è semplice, così come i processi di accesso ed elaborazione delle osservazioni. É qui che fa il suo ingresso il *SensorML* fornendo un framework per descrivere virtualmente ogni sensore così come il *processing* che potrebbe essere associato a questo.
Inoltre, il *SensorML*, si pone come un passo verso la preservazione di informazioni vitali richieste per la geolocalizzazione e l'elaborazione di osservazioni sia *real-time* che archiviate.
Fornisce dunque un mezzo standard grazie al quale le *capabilities* e le proprietà dei sensori e delle piattaforme possono essere facilmente pubblicate e scoperte.

3.1.4 *Dizionario e Ontologie*

Per migliorare la standardizzazione e l'univocità delle descrizioni fornite, il *SensorML*, ma anche l'**OGC** stesso, incoraggia l'uso di dizionari on-line o di ontologie gestite da autorità universalmente riconosciute. Un loro elenco è riportato nel documento *Definition identifier URNs in OGC namespace* [26].
Nella presente tesi è stata scelta l'autorità Environmental Data Coding Specification (EDCS) perché offre un dizionario on-line [2] molto ricco e facilmente consultabile.

3.1.4.1 *EDCS*

La EDCS fa parte del progetto Synthetic Environment Data Representation and Interchange Specification (SEDRIS) che:

- fornisce meccanismi per specificare in modo inequivocabile gli oggetti utilizzati per modellare concetti ambientali
- supporta la codifica e la comunicazione di informazioni qualitative e quantitative associate con gli ambienti fisici, sia reali che virtuali

- può essere utilizzata in congiunzione con qualsiasi modello di dati, struttura di dati o database per definire univocamente il tipo o le caratteristiche degli oggetti ambientali

3.2 52°NORTH

La *52°North* è un'organizzazione che fornisce un'implementazione delle specifiche fornite dall'iniziativa *SWE*. Questa non è l'unica alternativa presente. É stata scelta perché offre un'implementazione completa dei framework specificati nel SWE, il software sviluppato è ben documentato, ha un forum di supporto ed è già stato usato nel dipartimento di Ingegneria dell'Informazione dell'università di Pisa in merito al progetto *Cloud Sensor*.

3.3 CRBASIC

Il datalogger *CR1000* è programmabile usando il linguaggio CRBASIC. La parola è la contrazione di *CR* e di *Beginner's All purpose Symbolic Instruction Code (BASIC)*:

- *CR*, indica una classe di modelli di datalogger prodotti dalla **Campbell Scientific**
- *BASIC*, è un linguaggio di programmazione di alto livello

Il CRBASIC si presenta quindi come un linguaggio di programmazione basato sul BASIC e arricchito con istruzioni peculiari per l'interazione con il datalogger e l'hardware annesso.
Un programma generale per l'acquisizione di dati ha la seguente struttura:

Listing 1: Esempio di programma CRBASIC per l'acquisizione di due misure di temperatura

```
' Dichiarazione delle Variabili
Public T107_1
Public T107_2

' Dichiarazioni delle unita' di misura
Units T107_1 = Deg C
Units T107_2 = Deg C

' Definizione dei parametri archiviati ogni 5 minuti
DataTable(Tab1, True, -1)
        DataInterval(0, 5, Min, 10)

        Average(1, T107_1, FP2, False)
        Average(1, T107_2, FP2, False)
EndTable
```

```
' Corpo ciclico del programma che esegue le misure
BeginProg
        ' Il sensing e' effettuato una volta al minuto
        Scan(60, Sec, 1, 0)
                ' Acquisizione misura dal Termistore T107_1:
                Therm107(T107_1, 1, 15, 3, 0, _50Hz, 1.0, 0.0)
                ' Acquisizione misura dal Termistore T107_2:
                Therm107(T107_2, 1, 16, 3, 0, _50Hz, 1.0, 0.0)

                ' Chiamata allo statement DataTable per l'
                    immagazzinamento dei dati
                CallTable(Tab1)
        NextScan
EndProg
```

Le sezioni principali del programma sono:

- *Dichiarazioni*:
  - Variabili per contenere i valori acquisiti dai sensori
  - Unità di misura associate alle misure effettuate dai sensori
- *Immagazzinamento*:
  - Definizione delle tabelle che saranno salvate sul disco
  - Definizione dell'intervallo temporale in cui avviene l'immagazzinamento vero e proprio. Infatti, lo statement *DataTable* viene chiamato ad ogni misura effettuata mentre la memorizzazione sul disco avviene solo con l'intervallo temporale indicato dallo statement *DataInterval*. Facendo riferimento al Listing 1, le misure sono effettuate una volta ogni minuto mentre la memorizzazione solo una volta ogni 5 minuti.
  - Ai valori acquisiti è applicata una funzione di aggregazione. Sempre in riferimento al Listing 1, in 5 minuti sono prese 5 misure, sul disco viene salvato solo il valore medio tra questi.
- *Corpo del programma*:
  - É definito tra le parole chiave *BeginProgram* e *EndProgram*.
  - Contiene le istruzioni che accedono ai sensori per l'acquisizione dei dati. Esistono numerose istruzioni native che eseguono l'accesso diretto ai sensori supportati dal *CR1000*.
  - Per renderlo iterativo è usato lo statement *Scan()* che definisce l'intervallo di esecuzione del programma.
  - Per richiamare l'elaborazione e l'immagazzinamento dei dati è usato lo statement *CallTable()*.

4 ARCHITETTURA DEL PROGETTO

Questo capitolo descrive la struttura del servizio intermediario tra *CR1000* e utenti, progettato durante la tesi.

4.1 STRUTTURA AD ALTO LIVELLO

Nella Figura 22 è rappresentata la struttura ad alto livello che realizza il servizio intermediario tra stazioni meteorologiche, basate sull'acquisitore dati *CR1000*, e gli utenti che vogliono acquisirne le informazioni raccolte.

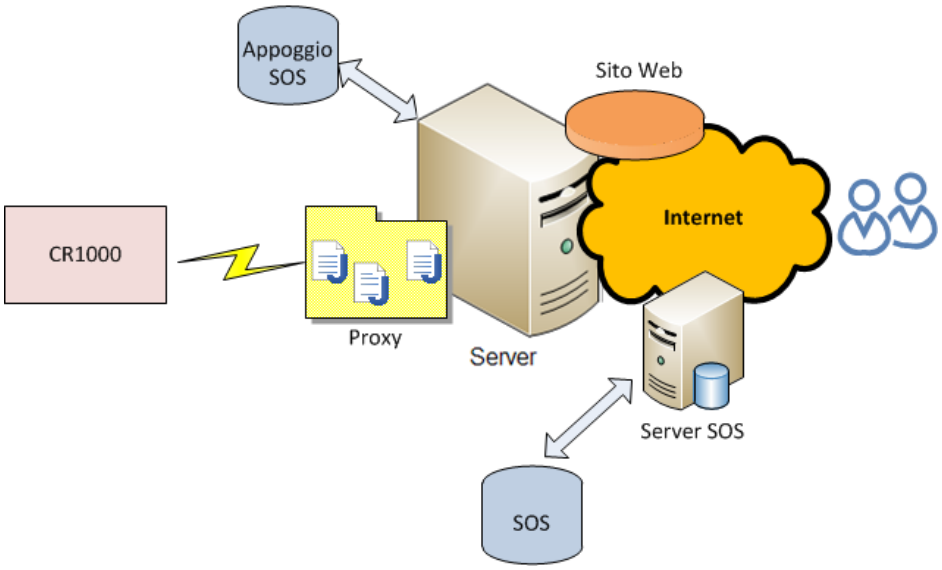

Figura 22: Schema ad Alto Livello del Progetto.

Dallo schema sono deducibili i componenti principali, ovvero il **Proxy** e il **Sito Web**.
Il **Proxy** adatta il *CR1000* al servizio SOS:
implementa l'acquisizione delle osservazioni dalla stazione meteorologica, compila e spedisce le richieste di *InsertObservation* popolando il database del SOS.
Il **Sito Web** si pone come un'interfaccia che permette agli utenti interessati di visualizzare e scaricare i dati raccolti dai sensori annessi alla Automatic Weather Station (AWS).

4.2 PROXY

Il **Proxy** è un'applicazione realizzata in Java che gira in *loop* continuo sul server su cui risiede. É stato necessario prevederlo principalmente per permettere l'interazione tra la stazione meteorologica e il servizio SOS.
Le informazioni ricevute dalla AWS sono salvate su file, uno per ogni sensore. Finita l'interazione con il *datalogger*, il **Proxy**, compila e inoltra le richieste di *InsertObservation* alla servlet del SOS.

4.2.1 *Servizi*

Nella Figura 23 è illustrata la struttura ad alto livello del **Proxy** che rende immediatamente evidenti gli altri servizi che questa applicazione implementa.

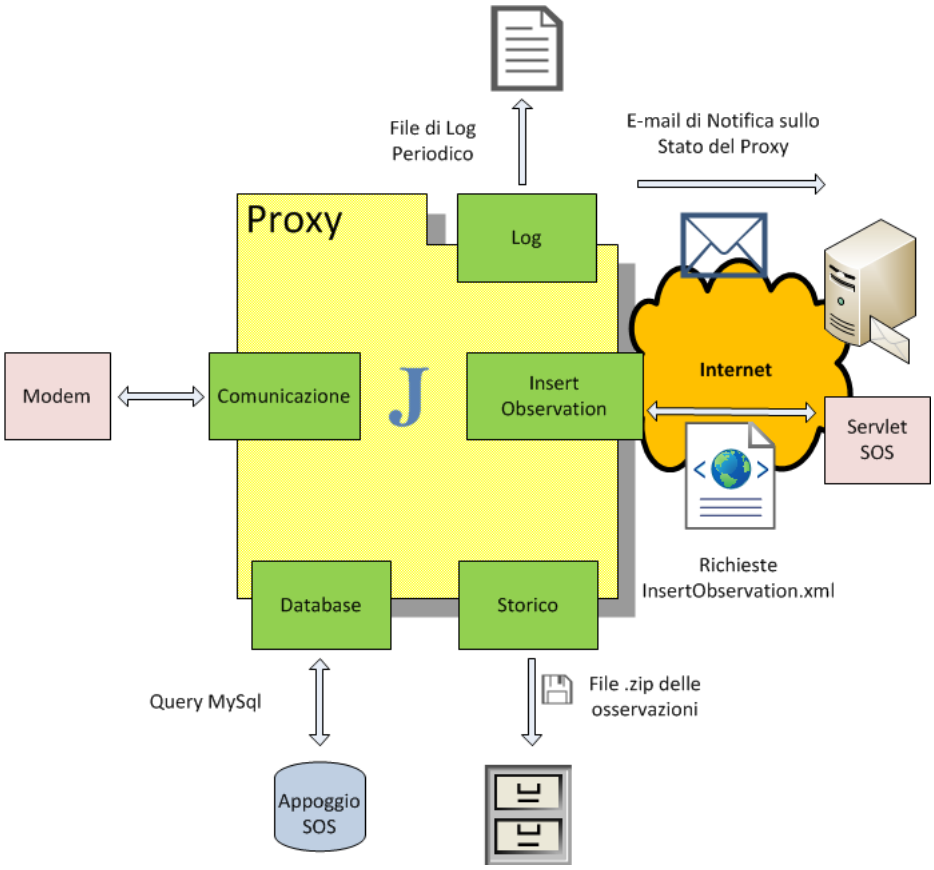

Figura 23: Schema ad Alto Livello del Proxy.

4.2.1.1 *Acquisizione Osservazioni dalla Stazione Meteorologica*

Le osservazioni sono inoltrate dal *CR1000* e sono salvate in appositi file, uno per ogni sensore annesso alla AWS.

4.2.1.2 *Inserimento delle Osservazioni nel Database del SOS*

La *52°North* implementa il servizio SOS tramite una Servlet Java. Per utilizzarla è necessario eseguire l'installazione [28] che termina con il *deployment*, del file *.war* generato, sul Tomcat. É possibile interagire con la servlet tramite richieste XML predefinite.
Per l'inserimento delle osservazioni nel database del SOS, il **Proxy**, compila e invia una richiesta di *InsertObservation*. La richiesta può:

- Andare a buon fine ⇒ il **Proxy** continua la sua esecuzione
- Generare un'eccezione ⇒ il **Proxy** la salva come *warning* sul proprio file di Log
- Fallire ⇒ il **Proxy** si salva l'osservazione non aggiunta al database e successivamente ritenta la richiesta.

4.2.1.3 *Log e Servizio di Notifica Stato all'Amministratore*

Per rendere l'applicazione del **Proxy** semplice da gestire e manutenere, è mantenuto un file di log ed è implementato un servizio di notifica del suo stato di esecuzione via e-mail.
Per quanto riguarda il file di log, l'applicazione si appoggia sul noto *log4j*, prodotto della *Apache Software Foundation*. Infatti, configurandolo opportunamente, l'applicazione, in modo automatico, è in grado di:

- Mantenere un file di log giornaliero sulle attività svolte dall'applicazione
- Impostare diversi livelli di gravità alle azioni/eventi che si svolgono durante l'esecuzione dell'applicazione

4.2.1.4 *Gestione dello Storico*

Per ragioni di robustezza viene mantenuto anche uno storico delle osservazioni sotto forma di file di testo compressi. Si tratta di ridondanza perché tutte le informazioni sono già immagazzinate nel database del SOS.
L'archiviazione dei dati avviene mensilmente e, i file compressi, sono scaricabili dal **Sito Web**.

4.3 SITO WEB

Il **Sito Web** è l'interfaccia con gli utenti:
permette di ricercare e visualizzare le osservazioni rendendo trasparente l'interazione con il SOS.
Notare nella Figura 22 che il **Sito Web** risiede sulla stessa macchina in cui risiede il **Proxy**. Questo perché, pur non interagendo direttamente, fanno riferimento a delle risorse in comune; ovvero:

- Il database interno, è acceduto in lettura dal **Proxy** e in scrittura dal **Sito Web**
- La cartella Storico, è popolata dal **Proxy** e il **Sito Web** rende visibile e scaricabile on-line il suo contenuto

Queste operazioni sarebbero eseguibili anche da remoto, ma è stata preferita un'architettura più compatta.

4.3.1 *Servizi*

Nella Figura 24 è illustrata la struttura ad alto livello del **Sito Web** che rende immediatamente evidenti gli altri servizi offerti agli utenti.

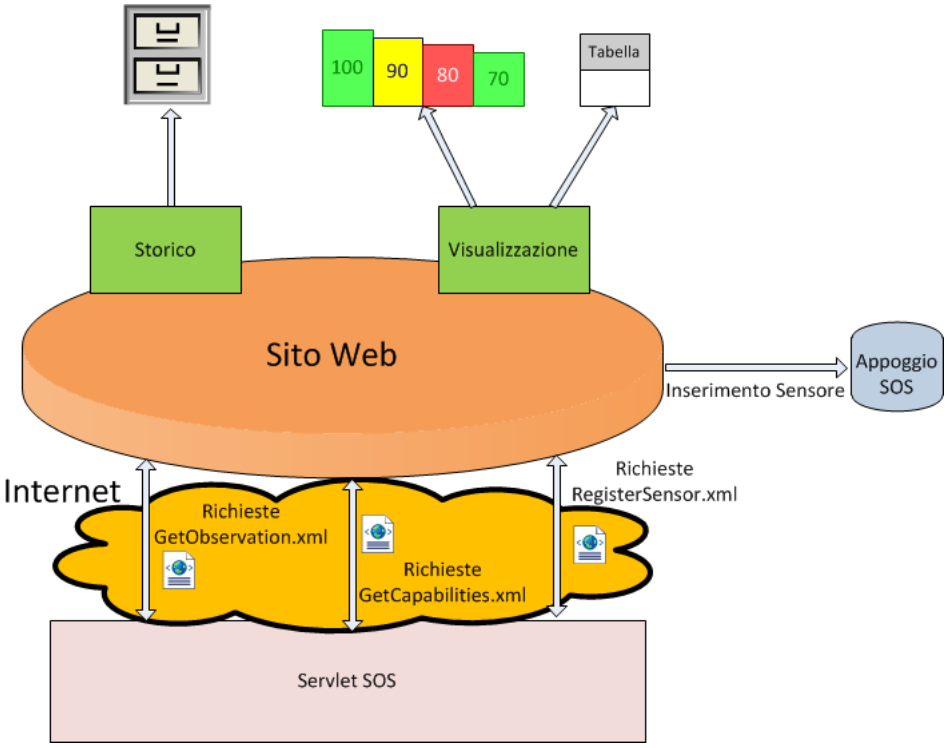

Figura 24: Schema ad Alto Livello del Sito Web.

4.3.1.1 *Registrazione di un Sensore*

Il **Sito Web** rende possibile, previa la compilazione di una form, la registrazione di un nuovo sensore alla stazione meteorologica. Contestualmente è prodotto anche un file, secondo le specifiche del SensorML, di descrizione del sensore e il popolamento del database interno *appoggio_SOS*.

4.3.1.2 *Visualizzazione delle Osservazioni*

Nella sezione *Grafici* del **Sito Web** sono presenti i grafici delle osservazioni dei sensori della AWS tenendo conto, come periodo temporale,

gli ultimi tre giorni.
Per quanto riguarda la visualizzazione tabellare dei dati, è possibile impostare dei filtri:

- Stazione Meteorologica
- Sensori
- Fenomeno
- Periodo temporale in cui l'osservazione è stata acquisita

Le informazioni per inizializzare i filtri sono recuperate tramite una richiesta di *getCapabilities* al SOS. I filtri impostati sono usati per compilare la richiesta di *GetObservation* sempre secondo la modalità di sostituzione dei campi variabili all'interno del *template* [vedi Listing 18].

4.3.1.3 *Storico*

Il **Sito Web** permette di accedere ai file compressi archiviati dal **Proxy** [vedi sottosottosezione 4.2.1.4] per scaricarli.

Parte II

IMPLEMENTAZIONE

In questa parte è eseguita un'analisi di fattibilità della tesi
e sono forniti i dettagli implementativi.

5

CONSUMO ENERGETICO DELLA STAZIONE METEOROLOGICA

In questo capitolo l'attenzione è puntata sul consumo energetico della stazione meteorologica dislocata sul ghiacciaio de La Mare. Con l'ausilio di formule e calcoli è dimostrata la fattibilità della comunicazione, a livello di consumi, della Automatic Weather Station (AWS).

5.1 MOTIVAZIONI

Il consumo energetico è uno dei vincoli più stringenti quando si ha che fare con sistemi che si devono auto-sostenere senza richiedere nè l'intervento umano nè l'attacco ad una rete elettrica. Solitamente, tali apparecchi, sono distribuiti in luoghi difficilmente accessibili all'uomo. Per queste ragioni è fondamentale tenere conto del consumo ad ogni fase della progettazione del sistema.

5.2 GSM VERSUS SATELLITE

Al fine di scegliere la migliore tecnica di comunicazione per il progetto, è stato effettuato un confronto tra le due tecnologie descritte nella sezione 2.6, evidenziando i pro e i contro di una soluzione rispetto all'altra.
L'analisi è riassunta in tabella:

Tabella 2: Tabella di confronto tra Fastrack Xtend e MiChroSat 2403.

	GSM	**Satellitare**	
	Fastrack Xtend	**MiChroSat 2403**	**Hughes 9502**
Consumo [mA]	idle→ 2.5 TX→ 400	sleep→ 15 TX→ 900	always-on→ 2.5 TX→ 1700
Rate TX (bps)	9600	2400	448k
Comandi AT	SI	SI	SI
Dati via seriale	SI	SI	NO
Copertura	La Mare→NO Ortles→Bassa	La Mare→SI Ortles→SI	La Mare→SI Ortles→SI
Costi [€]	Modem→≈ 130 TX dati→Bassa	Modem→≈ 1600 TX dati→ Alta	Modem→≈ 1000 TX dati→Medi-Alta

La copertura preclude la fattibilità del progetto. Per questo motivo è stata scelta la comunicazione satellitare nonostante i maggiori costi

e consumi.
I costi maggiori riguardano anche la comunicazione dei dati. Per *MiChroSat*, la tariffa è a tempo (*granularità dell'ordine dei minuti*) più un costo mensile fisso. Per *Hughes* sono presenti abbonamenti mensili basati sulla quantità (*granularità dell'ordine dei MB*) di dati trasmessi.
Nei successivi calcoli è stato considerato il modem *MiChroSat*, perchè il codice implementato per la comunicazione lato *CR1000* inoltra i dati al modem attraverso la porta seriale. Questa funzionalità non è supportata dal *Hughes* come evidenziato nella Tabella 2. Inoltre lo *Hughes*, usa un'antenna monodirezionale. Pur essendo il puntamento semplice, non dev'essere sottovalutato l'ambiente in cui la stazione meteorologica opera, ovvero:

- Lo spostamento del ghiacciaio, di circa dieci metri l'anno
- Le forti raffiche di vento
- Lo scioglimento della neve e del ghiaccio

I fattori sopra elencati portano ad un movimento dell'antenna con il conseguente deterioramento della connessione satellitare.

5.2.1 *Specifiche del modem MiChroSat 2403*

Per completezza sono riportate alcune specifiche del modem selezionato prima di procedere nei calcoli del consumo energetico della AWS con integrato il sistema di comunicazione.
Il *MiChroSat 2403* supporta sia connessioni modem-to-modem (Figura 25) che modem-to-Public Switched Telephone Network (PSTN)/GSM (Figura 26) fornendo un data rate di 2400bps. Quest'ultima modalità di funzionamento permette di risparmiare sull'acquisto di un secondo *MiChroSat 2403*.

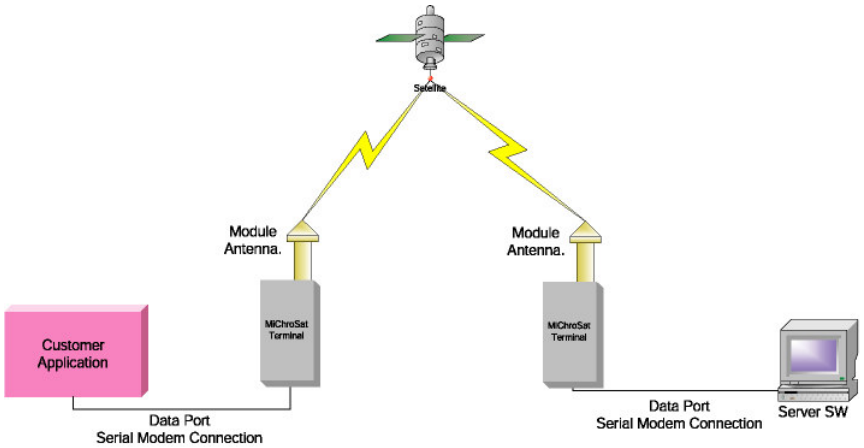

Figura 25: Schema connessione modem-to-modem del *MiChroSat 2403*.

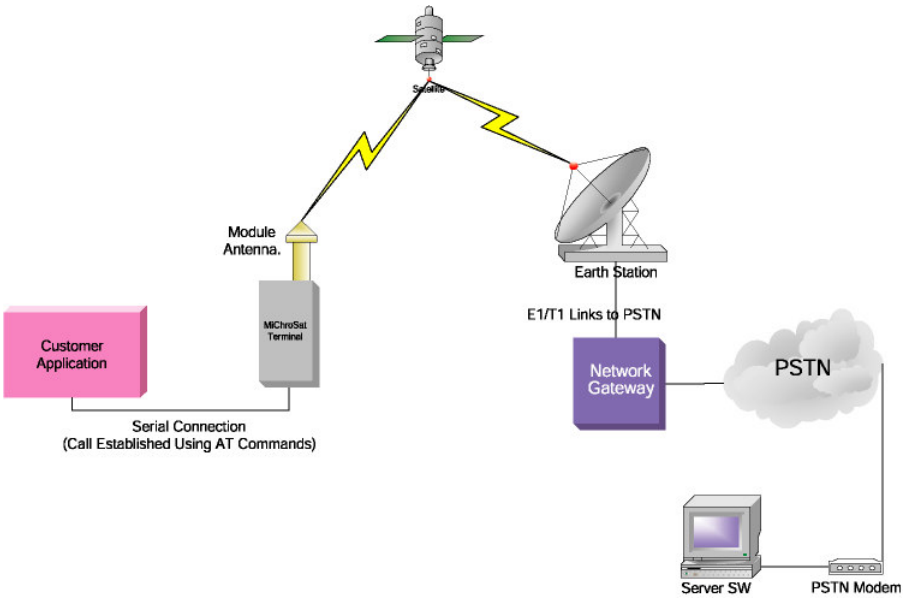

Figura 26: Schema connessione modem-to-PSTN del *MiChroSat 2403*.

Il modem presenta tre modalità di funzionamento: (*è considerata un'alimentazione di* 12V)

- *Sleep*, basso consumo energetico $\rightarrow$ 15mA
- *Idle*, medio consumo energetico $\rightarrow$ 150mA
- *Active*, alto consumo energetico $\rightarrow$ 900mA

Il *MiChroSat* può immagazzinare la configurazione, in modo da non richiedere una riconfigurazione in caso di spegnimento.
Il modem può essere acceso/spento automaticamente attraverso la *onboard intelligent controller interface* configurata attraverso il software *MiChroFace* incluso con l'acquisto dello stesso. Questa operazione può essere configurata in due modi:

- accensione/spegnimento in predeterminati momenti (*configurabili*)
- accensione/spegnimento eseguendo il sensing delle linee di controllo della porta seriale RS232 attraverso un'applicazione software (*External/Application RS232 Interrupt mode*)

In più ai comandi *Hayes* AT standard, sono forniti dei comandi specifici per il GSM e per l'Iridium.
L'antenna dev'essere posizionata in modo tale da garantire il Line Of Sight (LOS) con il satellite durante lo scambio dei dati. Visto il movimento dei satelliti è difficile prevedere in che parte del cielo si troveranno al momento della comunicazione, quindi l'antenna deve essere situata in un punto che non presenta ostruzioni verso l'alto.

5.3 ANALISI CONSUMI

Per analizzare i consumi della stazione meteorologica è stato fatto riferimento ad una nota tecnica della **Campbell Scientific** [18].

5.3.1 *Datasheet*

Per prima cosa sono state compilate delle tabelle con i dati relativi al consumo, prelevati dai corrispondenti datasheet, di tutti i moduli annessi al *CR1000*.
Per quanto riguarda i sensori:

Tabella 3: Caratteristiche utili per il calcolo dei consumi relativi ai sensori annessi alla stazione meteorologica in Val de La Mare.

	Idle (mA)	**Active (mA)**	**t_active (s)**
Termoigrometro HMP45C	0.0000	4.00	0.15
Termoigrometro CS215	0.0700	1.70	0.20
Nivometro SR50A	1.0000	250.00	1.00

I sensori non riportati in tabella hanno un consumo trascurabile.
Per quanto riguarda il modem:

Tabella 4: Caratteristiche utili per il calcolo dei consumi relativi al modem MiChroSat 2403.

	Sleep (mA)	**Idle (mA)**	**TX (mA)**	**Rate (bps)**	**Overhead (s)**
MiChroSat	15	150	900	2400	300

Il parametro *overhead* è **approssimativo** ed indica il tempo necessario per collegarsi e scollegarsi al satellite e le altre operazioni di settaggio della connessione.
Ed infine per il CR1000:

Tabella 5: Caratteristiche utili per il calcolo dei consumi relativi al CR1000.

	Idle (mA)	**Active 1Hz (mA)**	**Active 100Hz (mA)**	**Add for TX (mA)**	**t_active (s)**	**t_sampling (s)**
CR1000	0.6	1.0	16.2	10.0	5.0	60.0

Il parametro *Add for TX* indica il consumo aggiuntivo dovuto alla comunicazione.

5.3.2 *Dati*

Per ricavare il tempo di trasmissione necessario allo scambio dei dati, è necessario calcolare la quantità di byte scambiati giornalmente. Il codice CRBASIC salva una tabella per ogni tipologia di sensore. Nella Tabella 6 sono riassunte le dimensioni.

Tabella 6: Tabella riassuntiva della dimensione dei dati.

Tipologia	Sensore	Variabili	Valori (B)	TS (B)	RN (B)
Albedometro	LP PYRA05	RGup_Avg	2	8	4
		RGdown_Avg	2		
Anemometro	RM YOUNG	Vvento_S_WVT	2	8	4
		Vvento_U_WVT	2		
		Vvento_DU_WVT	2		
		Vvento_SDU_WVT	2		
Batteria	Wiring Panel	Batt_Volt	2	8	4
Nivometro	SR50A	DT	2	8	4
Pirgeometro	CGR3_upward	IRup_Avg	2	8	4
		IRupc_Avg	2		
	CGR3_downward	IRdown_Avg	2		
		IRdownc_Avg	2		
Termistore	T107-1	T107_1_Avg	2	8	4
	T107-2	T107_2_Avg	2		
	Wiring Panel	TCR1000	2		
Termoigrometro	CS215	AirTC_Avg	2	8	4
		RH_Avg	2		
	HMP45C	Thmp45_Avg	4		
		URhmp45_Avg	4		

Dove:

- Time Stamp (TS), indica il momento in cui l'osservazione è stata immagazzinata
- Record Number (RN), indica il contatore del record immagazzinato nella tabella

Il programma attuale ha due diversi rate di immagazzinamento:

- Un immagazzinamento ogni ora per la Batteria e per il Nivometro $\Rightarrow 1440/60 = 24$ osservazioni archiviate giornalmente

- Un immagazzinamento ogni cinque minuti per gli altri sensori $\Rightarrow 1440/5 = 288$ osservazioni archiviate giornalmente

Quindi, giornalmente, sono prodotti:

$$[288 * (38 + 40 + 20)] + [24 * (4 + 16 + 8)] = 28896[B]$$

5.3.3 *Formule*

La seguente formula serve per calcolare la *current drain*[1] media e viene applicata ad ogni modulo del sistema:

$$\frac{(t_{active} * active) + [(t_{sampling} - t_{active}) * idle]}{t_{sampling}} \quad (1)$$

Nell'esecuzione dei calcoli considerare:

- t_{active}, il tempo in cui l'apparecchio è in modalità attiva
- $active$, parametro presente nelle tabelle dei moduli per indicare la *current drain* in modalità *active*.
- $t_{sampling} = 60$, ovvero il sensing avviene una volta al minuto
- $idle$, parametro presente nelle tabelle dei moduli per indicare la *current drain* in modalità *idle*.

Applicando la formula ai sensori e al CR1000 sono ottenuti i seguenti valori:

Tabella 7: Consumi dei sensori e del CR1000.

	Current Drain Avg (mA)
Termoigrometro HMP45C	0.010
Termoigrometro CS215	0.074
Nivometro	5.150
CR1000	1.383

Per quanto riguarda il *CR1000* è stato considerato come *active* il valore $10mA$. Infatti, come riportato nella nota tecnica della **Campbell Scientific** [18], una stazione meteorologica basata su *CR1000* ha una *active current drain* media di $10mA$.
Per quanto concerne la comunicazione è necessario calcolare il tempo di trasmissione:

$$t_{TX} = Dati_{Giornalieri}/RateTX[B]$$

1 indica la corrente *consumata* da un apparecchio elettronico

Inoltre, al valore di consumo in modalità di trasmissione del modem, vanno aggiunti 10mA relativi al *CR1000*, come riportato nella Tabella 5.
Visto che il modem può trovarsi in tre differenti modalità, la formula 1 va estesa:

$$\frac{(overhead * idle) + (t_{TX} * TX) + [(86400 - t_{TX} - overhead) * sleep]}{86400}$$

Dove 86400 indicano i secondi che compongono un giorno; infatti, è considerata una comunicazione al giorno. Eseguendo i calcoli:

Tabella 8: Consumi del Modem MiChroSat 2403.

	Current Drain Avg (mA)
MiChroSat 2403	16.467

5.4 SISTEMA DI ALIMENTAZIONE

Per capire se il sistema di alimentazione è in grado di supportare i consumi appena descritti, che ammontano a 23.08mA, è necessario fare delle considerazioni sul sistema di alimentazione.
Come visto nella sezione 2.5, il sistema di alimentazione è composto da una batteria ricaricabile con capacità di 24A/h e un pannello solare da 20W con una corrente di picco di 1.170A.

5.4.1 *Batteria Ricaricabile*

Per sapere il valore di capacità richiesta alla batteria è necessario applicare la seguente formula:

$$Capacita\_richiesta\_batteria = \frac{CD\_Stazione_{Avg} * Reserve\_Time}{0.8} \tag{2}$$

Dove:

- $CD\_Stazione_{Avg}$ è la *Current Drain* media della stazione meteorologica, riportato in *Ampere*
- Il $Reserve\_Time$ è un parametro prelevato dalla Tabella 9 che tiene conto della relazione tra latitudine del sito e requisiti di capacità della batteria
- 0.8 serve per considerare una scarica della batteria del 80% [*worst case*]

Tabella 9: Tabella che riporta l'associazione tra Latitudine del sito e Reserve Time [18].

Latitudine Sito (°)	**Reserve Time Raccomandato (h)**
[0; 30[	[144; 168]
[30; 50[	[288; 336]
[50; 60[	[432]
Regioni Polari	[8.760]

Vista la latitudine di 46°26′ del sito sul ghiacciaio de La Mare, va considerato un $Reserve\_Time$ di 336h, mentre per la $CD\_Stazione_{Avg}$ è stato ricavato 0.0231A. Quindi, sostituendo i valori nella formula 2:

$$\frac{0.0231 * 336}{0.8} = 9.702[Ah]$$

Dunque la capacità richiesta alla batteria per alimentare il sito è di almeno 9.702Ah. Per ragioni precauzionali è sempre bene considerare un margine di sicurezza additivo del 20%, quindi la $Capacita\_richiesta\_batteria$ ammonta a 11.642Ah.

5.4.2 *Pannello Solare*

Per sapere se il pannello solare in dotazione è sufficiente per supportare la ricarica della stazione meteorologica è necessario innanzitutto calcolare gli *Ah* medi della AWS giornalmente:

$$Ah/giorno = CD\_Stazione_{Avg} * 24 \tag{3}$$

Dove 24 sono le ore che compongono un giorno. Quindi, sostituendo i valori nella formula 3:

$$0.0231 * 24 = 0.5544[Ah/day]$$

Infine è necessario calcolare la corrente richiesta al pannello solare:

$$Corrente\_SP = \frac{Ah/giorno_{Sistema} * 1.2}{h\_luce\_giornaliere} \tag{4}$$

Dove per $h\_luce\_giornaliere$ sono intese le ore al giorno in cui il pannello solare può godere della luce solare ed sono state considerate 5 ore al giorno. Quindi, sostituendo i valori nella formula 4:

$$\frac{0.5544 * 1.2}{5} = 0.1331[A]$$

Dunque il pannello solare deve erogare almeno una corrente di 0.133A per ricaricare la batteria. Per ragioni precauzionali è sempre bene considerare un margine di sicurezza additivo del 20%, quindi la Corrente_SP ammonta a 0.160A.

5.4.3 *Conclusioni*

Nella Tabella 10 sono riassunti i calcoli eseguiti. É evidente che, anche considerando i margini di sicurezza, il sistema di alimentazione è sufficiente a supportare il consumo della stazione meteorologica pur dotandola di comunicazione satellitare.

Tabella 10: Tabella riassuntiva sui consumi della stazione meteorologica in Val de La Mare.

	Stazione Meteo La Mare
Dati Giornalieri (B)	28896
Reserve Time (h)	336
Current Drain Avg [A)	0.0231
Ah/giorno (Ah/day)	0.5544
Capacità Rich. Batteria (Ah)	9.7020
Capacità Rich. Batteria Sic. (Ah)	11.6420
Capacità Batteria (Ah)	24
Corrente SP (A)	0.1330
Corrente SP Sic. (A)	0.1600
Corrente di Picco SP (A)	1.1700

Come è evidenziato nella sottosezione 6.5.4 il modem viene acceso solo quando è necessario comunicare. Questo comporta un notevole risparmio energetico [14.93mA] in quanto il modem non sta più in modalità *sleep*. Previa questa considerazione, il consumo medio giornaliero della AWS ammonta a circa 8.154mA.

6 DETTAGLI IMPLEMENTATIVI

In questo capitolo sono illustrate le tecnologie effettivamente usate a livello di software ed è presentata una descrizione dettagliata dei meccanismi implementati.

6.1 TECNOLOGIE USATE

Per l'implementazione del **Sito Web** è stata scelta la triplette Apache, My Structured Query Language (MySQL) e PHP:Hypertext Preprocessor (PHP) mentre il **Proxy** è stato sviluppato come un'applicazione Java.
Il sistema operativo adoperato è *Ubuntu 11.10* ed è fatto girare su una macchina virtuale. L'emulatore è *VirtualBox* della **Oracle**. Entrambi sono scaricabili gratuitamente.

6.1.1 *Apache - MySQL - PHP*

É stata scelta la tripletta Apache, MySQL e PHP perché sono software *opensource* nonché la tecnologia più diffusa nell'implementazione di siti web che richiedono l'interazione con un database.
Le versioni installate sono:

- Apache → pacchetto *apache2*
- MySQL → pacchetto *mysql-server*
- PHP → pacchetto *php5*

6.1.2 *JavaScript - HTML - DOM*

Per il **Sito Web** sono stati usati anche:

- JavaScript per i controlli lato cliente
- HyperText Markup Language (HTML) per la scrittura delle pagine statiche che compongono il sito
- Document Object Model (DOM) per l'analisi e l'estrapolazione delle informazioni dai file XML scambiati con la servlet che implementa il servizio SOS

6.1.3 *Java Eclipse*

É stato scelto il Java in quanto è un linguaggio estremamente portatile e dotato di molte API che lo rendono flessibile e adatto alle funzionalità che il **Proxy** deve implementare.
L'Integrated Development Environment (IDE) usato è **Eclipse Java EE IDE for Web Develpers** versione *Indigo Service Release 1* [10].

6.2 DATABASE APPOGGIO_SOS

Il sistema incorpora un database chiamato *appoggio_SOS* [vedi Figura 22] che contiene informazioni riguardo ai sensori annessi alle stazioni meteorologiche e ai fenomeni che misurano.
Questo database non è strettamente necessario, la sua presenza è tuttavia motivata da una riduzione della complessità del **Proxy**. Infatti, in fase di compilazione della richiesta di *InsertObservation*, il **Proxy**, recupera le informazioni necessarie dal database *appoggio_SOS* anziché:

- Eseguire una richiesta preventiva di *GetCapabilities* al SOS per acquisire i nomi dei sensori a lui registrati.
- Per ogni sensore registrato eseguite una richiesta di *DescribeSensor* per recuperare i campi necessari [Unità di misura, posizione...]

La procedura descritta pare lineare e pulita. Va considerato però che i sensori registrati al SOS saranno molti di più rispetto a quelli appartenenti alla stazione meteorologica di interesse e che i risultati sono file XML la cui gestione non è immediata.
Quindi è stata preferita una perdita di elasticità per garantire al software una maggiore semplicità, leggerezza e scalabilità.

6.2.1 *Utenti Coinvolti*

Nel diagramma dei casi d'uso sono mostrati i due attori che interagiscono con il database:

- **Proxy**:
  - Esegue query per ottenere informazioni
  - Per ragioni di concorrenza esegue *Lock* in lettura sulle tabelle prima di accedervi
- **Sito Web**
  - Esegue l'inserimento dei nuovi sensori registrati al servizio SOS

– Per ragioni di concorrenza esegue *Lock* in scrittura sulle tabelle prima di accedervi

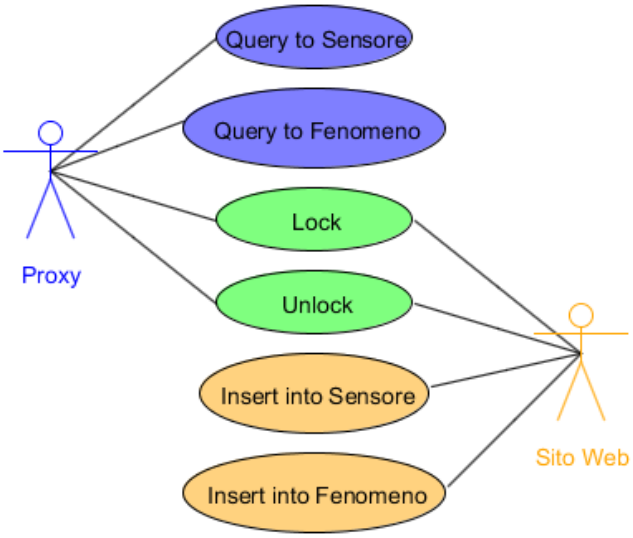

Figura 27: Diagramma UML dei casi d'uso del database.

6.2.2 *Modello Entity-Relationship*

Il primo step nella progettazione della base di dati è stata la costruzione del modello Entity-Relationship (E-R). In questo modo sono state individuare le entità e le relazioni necessarie con le rispettive cardinalità:

- (1,N):
  - (1) un sensore deve misurare almeno un fenomeno
  - (N) un sensore può misurare più fenomeni
- (1,N):
  - (1) un fenomeno dev'essere misurato da almeno un sensore
  - (N) un fenomeno può essere misurato da uno o più sensori

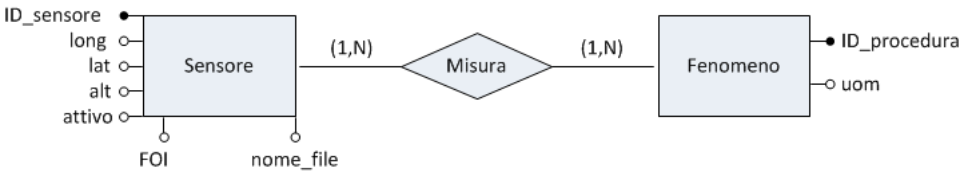

Figura 28: Schema del modello E-R del database.

6.2.3 *Dizionario dei Dati*

Il dizionario dei dati è composto dalla tabella delle entità e dalla tabella delle relazioni. Serve per dare una descrizione immediata e univoca dell'entità/relazione e dei campi che la compongono.

6.2.3.1 *Tabelle dell'Entità*

Il database è molto semplice ed è composto da due sole entità:

- Sensore
- Fenomeno

Tabella 11: Tabella che descrive le entità coinvolte nel database

Entità	Descrizione	Attributi	Identificatore
Sensore	Descrive i campi che riguardano il sensore	**ID_sensore**, long, lat, alt, attivo, nome_file, FOI	ID_sensore
Fenomeno	Descrive i campi che riguardano i componenti del sensore	**ID_fenomeno**, uom	ID_fenomeno

6.2.3.2 *Tabelle delle Relazioni*

Tra le due entitè sussiste una relazione: un sensore deve compiere la *misura* di uno o più fenomeni.

Tabella 12: Tabella che descrive le relazioni coinvolte nel database

Relazione	Descrizione	Attributi	Identificatore
Misura	Indica la relazione di misura tra il Sensore e i corrispondenti Fenomeni		

6.2.4 *Modello Relazionale*

Nel passaggio dallo schema E-R al modello relazionale la chiave della tabella *Sensore* è andata a far parte della chiave primaria della tabella *Fenomeno*.

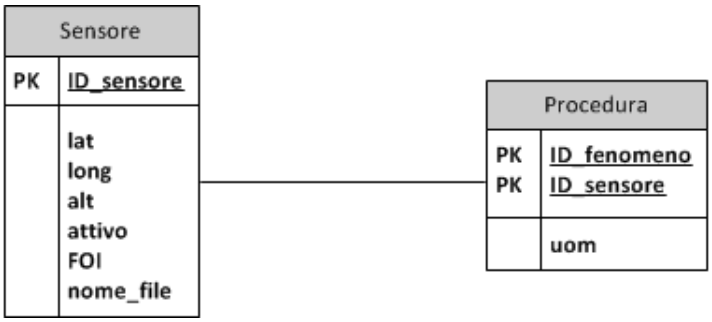

Figura 29: Modello relazionale del database interno *appoggio_SOS*.

6.2.5 *Implementazione del Database in MySQL*

Il database è semplice e, di conseguenza, anche lo script per generarlo:

Listing 2: Script MySQL per la creazione del database appoggio_SOS

```
-- Creazione del DB
create database appoggio_SOS;

-- Selezione DB
use appoggio_SOS;

-- Creazione tabella Sensore
create table Sensore(
        ID_sensore varchar(70) not null,
        nome_file varchar(70) not null,
        attivo tinyint(1) not null default 1,
        lat int not null default 46.26,
        lon int not null default 10.38,
        alt int not null default 2970,
        FOI varchar(20) not null default 'CR1000-9662',
        primary key (ID_sensore)
);

--Creazione tabella Fenomeno
create table Fenomeno(
        ID_fenomeno varchar(70) not null,
        ID_sensore varchar(70) not null,
        uom varchar(70) not null,
        primary key (ID_fenomeno, ID_sensore)
);
```

Da notare come i valori di default siano impostati in riferimento alle caratteristiche della stazione meteorologica sul ghiacciaio de La Mare.

6.2.6 *Concorrenza*

Come appare evidente dal diagramma dei casi d'uso nella Figura 27, al database accedono due diversi *utenti*. Per evitare situazioni di inconsistenza è dunque necessario gestire la concorrenza. Il meccanismo implementato è molto semplice e consiste nell'acquisire il *lock* sulle tabelle prima del loro utilizzo; in particolare:

- Lock in Lettura, per eseguire query
- Lock in Scrittura, per eseguire inserimenti nelle tabelle

A livello di pseudo codice:

Listing 3: Pseudo codice per la gestione dei lock

```
lock tables nome_tabella read | write
-- operazione
unlock tables
```

6.3 PROXY

In questa sezione è analizzato in dettaglio l'implementazione del **Proxy** nei suoi vari moduli.
L'architettura del progetto Java a livello di package è illustrata nel diagramma UML dei package in Figura 30. Un package è un raggruppamento di elementi e fornisce loro un *namespace* [1].

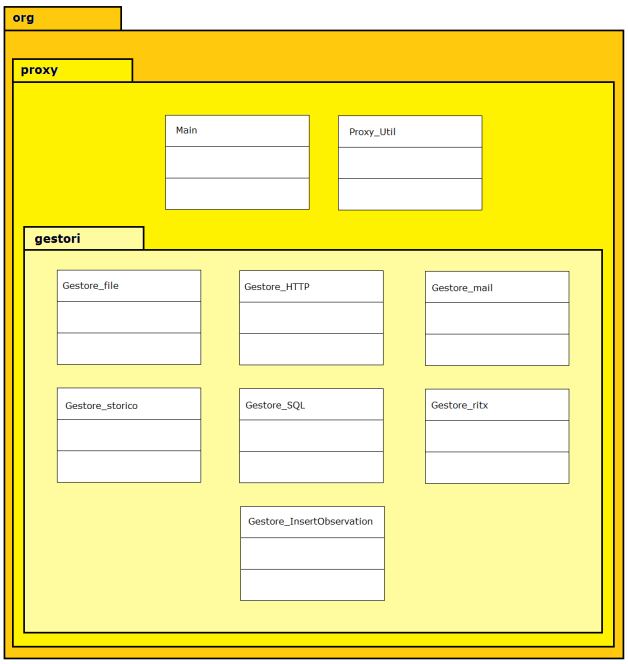

Figura 30: Diagramma UML dei package del **Proxy**.

1 entità che permette di identificare ogni elemento con il suo nome

6.3.1 *Archiviazione dei File*

Come accennato nella sottosottosezione 4.2.1.4, il **Proxy**, si occupa della gestione dello Storico; in altre parole dell'archiviazione dei dati acquisiti dalla stazione meteorologica.
La procedura di archiviazione, per default, avviene ogni primo giorno del mese. Tramite la costante *GIORNO_STORICO* è possibile impostare un altro giorno.
La compressione è eseguita grazie all'API *java.util.zip* e i file hanno un nome che rispetta il seguente formato:
st-<ID-sensore> - dd-mm-yyyy.zip.

6.3.2 *Interazione con il Database appoggio_SOS*

Il **Proxy** interagisce con il database interno per recuperare:

- I nomi dei file in cui sono salvate le osservazioni in seguito alla comunicazione con il *CR1000*
- Alcune delle informazioni necessarie per compilare la richiesta di *InsertObservation*

Java DataBase Connectivity (JDBC) mette a disposizione una libreria di classi Java per interfacciare l'accesso ai database che usano lo standard Structured Query Language (SQL), con l'obiettivo di fornire un'uniformità di accesso per un insieme vasto di prodotti DataBase Management System (DBMS). Attraverso JDBC è possibile richiamare comandi SQL direttamente. La Figura 31 illustra quanto detto.

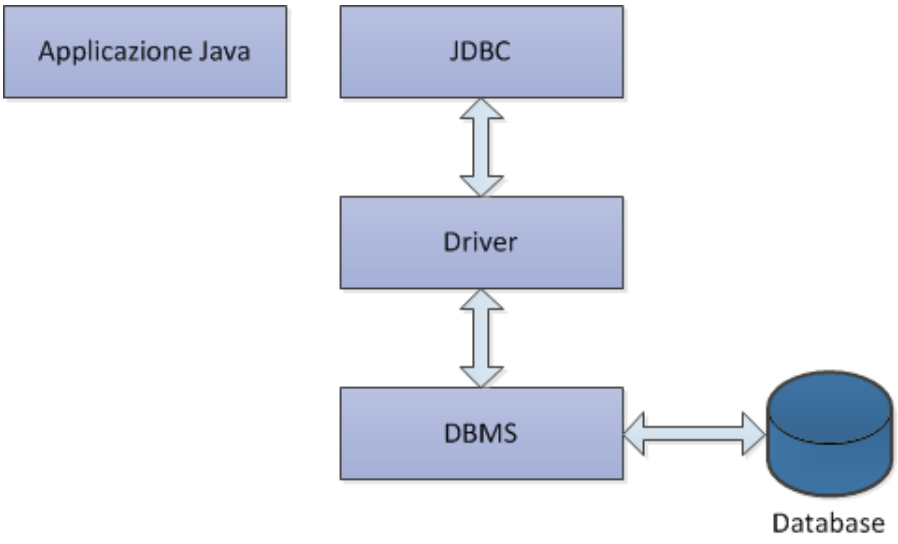

Figura 31: Diagramma che mostra come Java interagisce con i Database.

Per potere usare JDBC è necessario includere nel *build path* del progetto il Java ARchive (JAR) **mysql-connector-java-<VERS>-bin.jar**. In-

fine usare la classe *DriverManager* che tiene traccia dei driver disponibili e li gestisce stabilendo la connessione tra un driver e un particolare database:

Listing 4: Caricamento del driver JDBC

```
// ...
try
{
        Class.forName("com.mysql.jdbc.Driver").newInstance();
}
catch(ClassNotFoundException e)
{
        System.err.println("Eccezione ClassNotFoundException: " +
             e.getMessage());
}
```

A questo punto eseguire una query risulta immediato:

Listing 5: Connessione al database ed esecuzione di una query

```
// ...
Connection db;
Statment stmt;
ResultSet rs;
ResultSetMetaData rsmd;

try
{
        // Connessione
        db = DriverManager.getConnection("jdbc:mysql://localhost
            /<DB>?user=<USER>&password=<PSW>");
        // Esecuzione Query
        rs = stmt.execute(<QUERY>);
        rsmd = rs.getMetaData();
}
catch(SQLException e)
{
        System.err.println("Eccezione SQLException: " + e.
            getMessage());
}
```

6.3.3 *Compilazione della Richiesta di InsertObservation*

Per comporre la richiesta necessaria all'inserimento delle osservazioni, il **Proxy**, dispone di un *template* [2] della *InsertObservation*. I campi variabili sono indicati con dei tag prefissati di modo che possano essere individuati e sostituiti facilmente. Come descritto nella Tabella 13 alcuni campi sono recuperati dal file scritto in seguito alla comunicazione con il *CR1000* mentre altri dal database interno.

2 file con una struttura fissa predefinita (vedere Listing 17)

Tabella 13: Tabella che contiene gli elementi che saranno sostituiti nel *template* della *InsertObservation*.

Campo	Recuperato da...
<ID_Sensore>	Database interno *appoggio_SOS*
<Elenco_Fenomeni>	Database interno *appoggio_SOS*
<FOI>	Database interno *appoggio_SOS*
<lan>	Database interno *appoggio_SOS*
<lon>	Database interno *appoggio_SOS*
<Elenco_Campi>	Database interno *appoggio_SOS*
<TS>	File Proveniente dal *CR1000*
<Elenco_Valori>	File Proveniente dal *CR1000*

6.3.4 *Interazione con il Servizio SOS*

Il **Proxy** interagisce con la servlet che implementa il SOS solo per l'inserimento delle osservazioni. Una volta compilata la richiesta di *InsertObservation* (vedi sottosezione 6.3.2), questa viene inoltrata alla servlet passandola come argomento di una POST HyperText Transfer Protocol (HTTP). Se la POST va a buon fine, allora il **Proxy** si limita a controllare che la risposta non contenga l'elemento *<ows:ExceptionText></ows:ExceptionText>*:
se presente, il suo valore viene salvato come *Warning* nel file di Log.
Se invece la POST fallisce, la causa è l'irrangiungibilità del server che ospita il SOS. Ovviamente nell'ipotesi che SOS e **Proxy** girino su due macchine distinte. Per la gestione di questo caso vedere la sottosottosezione 6.3.4.1.
Il *CR1000* supporta il protocollo HTTP, quindi perché non fargli eseguire direttamente le richieste XML al SOS? La risposta è semplice:
il tempo di vita della stazione meteorologica si ridurrebbe drasticamente e il costo della trasmissione aumenterebbe vertiginosamente.
Tale soluzioni, infatti, richiederebbe un uso intensivo della comunicazione e la trasmissione di una quantità di dati molto maggiore che aumenterebbe la durata della connessione satellitare. Inoltre, la gestione di eventuali eccezioni, risulterebbe complessa e pesante per il *CR1000*. In tal caso le possibilità sarebbero due:

- Servizio Real-Time, il *CR1000* esegue richieste di *InsertObservation* ad ogni osservazione acquisita.
- Servizio Giornaliero, il *CR1000* esegue le richieste di *InsertObservation* giornalmente.

La soluzione prevista, invece, prevede l'inoltro dei dati acquisiti dalla stazione meteorologica al **Proxy**, ottenendo i seguenti vantaggi:

- Minimizzazione del tempo di comunicazione del *CR1000*
- Minimizzazione delle modifiche d'apportare al programma che gira sul *CR1000*

6.3.4.1 *Gestione delle Ritrasmissioni*

In caso di irraggiungibilità del SOS è stato previsto un meccanismo di ritrasmissione delle richieste di *InsertObservation*, di modo che tutte le osservazioni siano inserite nel database del SOS.
Il meccanismo prevede il salvataggio delle osservazioni e delle corrispondenti informazioni, di modo da non ri-compiere un interrogazione al database interno, in un file di testo. La ritrasmissione è tentata il giorno successivo in seguito alla comunicazione con il *CR1000* e prima di inviare le nuove osservazioni. Dal file saranno eliminate, volta volta, le osservazioni la cui trasmissione è andata a buon fine.
Unica nota interessante è l'inizializzazione del membro statico *n_righe* della classe **Gestione_ritx**. Questa è inizializzata dal valore di ritorno del metodo *contaRighe()* che conta le righe del file in cui sono inserite le osservazioni da ritrasmettere[3]. In questo modo sono evitate situazioni erronee dovute ad un'interruzione imprevista del **Proxy** con richieste di *InsertObservation* fallite da ritrasmettere.

6.3.5 *File di Log*

Come accennato nella sottosottosezione 4.2.1.3, il **Proxy**, mantiene un file di log giornaliero appoggiandosi sul *log4j*. Questo strumento è molto semplice e flessibile. I passi da compiere sono:

- Aggiungere al *build path* del progetto del **Proxy** il JAR **log4j-<VERS>.jar**
- Creare il file di configurazione *log4j.properties* (vedi Listing 7)
- Istanziare un oggetto di tipo *Logger* per ogni classe che compone il progetto:

Listing 6: Definizione del logger per una classe

```
private static final Logger logger = Logger.getLogger(<
    NomeClasse>.class);
```

Il file di configurazione adoperato per il **Proxy** è:

3 Ogni osservazione risiede su una riga

Listing 7: File di configurazione del log4j del Proxy

```
log4j.rootCategory=DEBUG, APPENDER_DAILY, APPENDER_CONSOLE

#APPENDER_DAILY
log4j.appender.APPENDER_DAILY=org.apache.log4j.
    DailyRollingFileAppender
log4j.appender.APPENDER_DAILY.File=./Logs/Proxy.log
log4j.appender.APPENDER_DAILY.Append=true
log4j.appender.APPENDER_DAILY.DatePattern='.'yyyy-MM-dd
log4j.appender.APPENDER_DAILY.layout=org.apache.log4j.
    PatternLayout
log4j.appender.APPENDER_DAILY.layout.ConversionPattern=%d{dd-MM-
    yyyy HH:mm:ss} %p %t %c - %m%n

#APPENDER_CONSOLE
log4j.appender.APPENDER_CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.APPENDER_CONSOLE.layout=org.apache.log4j.
    PatternLayout
log4j.appender.APPENDER_CONSOLE.layout.ConversionPattern=%p %t -
    %m%n
```

Nella prima riga è definito il livello minimo di errore che viene considerato dal *Logger* e i flussi di output dove il *Logger* riporterà gli errori. Sono definiti due flussi di output:

- APPENDER_DAILY, trascrive gli errori su un file. Il rollover avviene a mezzanotte di ogni giorno su un file dal nome *Proxy.log.<DATA>*
- APPENDER_CONSOLE, visualizza gli errori sulla console

6.3.6 *Servizio di Notifica all'Amministratore*

Il servizio di Notifica serve per aggiornare l'amministratore del servizio sullo stato di esecuzione del **Proxy**. L'e-mail è spedita in seguito alla comunicazione con il *CR1000* ed ha allegato il file di Log odierno. Se l'amministratore non riceve l'e-mail può dedurre che il **Proxy** sia crashato. É possibile impostare un elenco di indirizzi da contattare impostando la costante *MAIL_TO_ADDRESSES*.
In ambiente Java, per mandare messaggi di posta elettronica, è necessario scaricare ed includere nel *build path* i JAR **mail-<VERS>.jar** e **activation.jar**. Il protocollo usato è il Simple Mail Transfer Protocol (SMTP) che si occupa dell'invio dell'e-mail; infatti, quando viene inviato un messaggio di posta elettronica, questo è inoltrato ad un SMTP mail server il quale si occupa di mandarlo al mail server di destinazione:

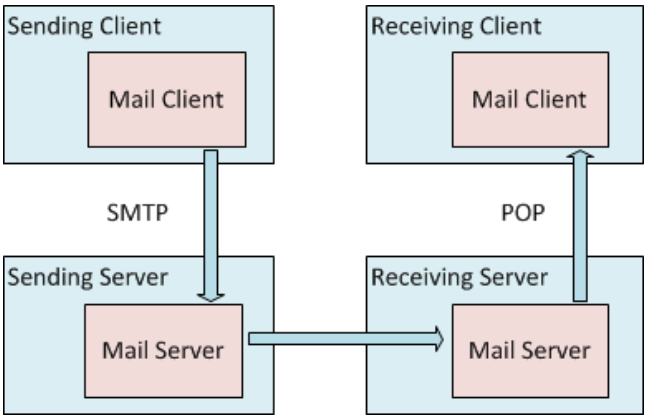

Figura 32: Schema a blocchi sul funzionamento del servizio di posta elettronica.

Il codice per aggiungere il contenuto al messaggio è semplice:

Listing 8: Aggiunta del testo e dell'allegato al messaggio di posta elettronica

```
// Creazione del contenuto della mail
Multipart multipart = new MimeMultipart();

// Prima parte: testo
BodyPart messageBodyPart = new MimeBodyPart();
messageBodyPart.setText(Proxy_Util.MAIL_TESTO);
multipart.addBodyPart(messageBodyPart);

// Seconda parte: allegato
messageBodyPart = new MimeBodyPart();
FileDataSource fds = new FileDataSource(Proxy_Util.PATH_LOG);
messageBodyPart.setDataHandler(new DataHandler(fds));
messageBodyPart.setFileName(fds.getName());
multipart.addBodyPart(messageBodyPart);

// Aggiunta del contenuto al messaggio
msg.setContent(multipart);
```

É creato un oggetto *Multipart* in cui prima è aggiunto il testo e poi l'allegato; infine, il tutto, è additato all'e-mail stessa.

6.3.7 *File di utilità: Proxy_Util*

Per rendere il codice facilmente configurabile è prevista una classe che contiene tutte le costanti usate dal **Proxy**. In questo modo gli utenti possono agire sui parametri dell'applicazione, come le credenziali di accesso al server mail, senza dover mettere mano al codice dell'applicazione. Questo file è adeguatamente commentato, la sua modifica è semplice ed intuitiva.

6.3.8 *Documentazione*

Tutto il codice sviluppato all'interno dell'applicazione Java del **Proxy** è stato documentato con il tool *doxygen* [9]. Questo strumento è scaricabile in modo gratuito e permette, previa la compilazione del file di configurazione, la generazione automatica della documentazione del software. Unica accortezza:
i commenti devono rispettare una precisa sintassi per essere riconosciuti dal *doxygen*.
Per generare i grafici delle classi è necessario installare il tool *graphviz*[11].

6.3.9 *Esempio di Funzionamento - Inserimento Osservazioni nel database SOS*

Per chiarezza nella Figura 33 è riportato il diagramma UML di sequenza per l'inserimento di un'osservazione nel database del SOS. É supposto un funzionamento corretto, quindi senza errori nelle interazioni con il database interno *appoggio_SOS* e nell'esecuzione della post HTTP.

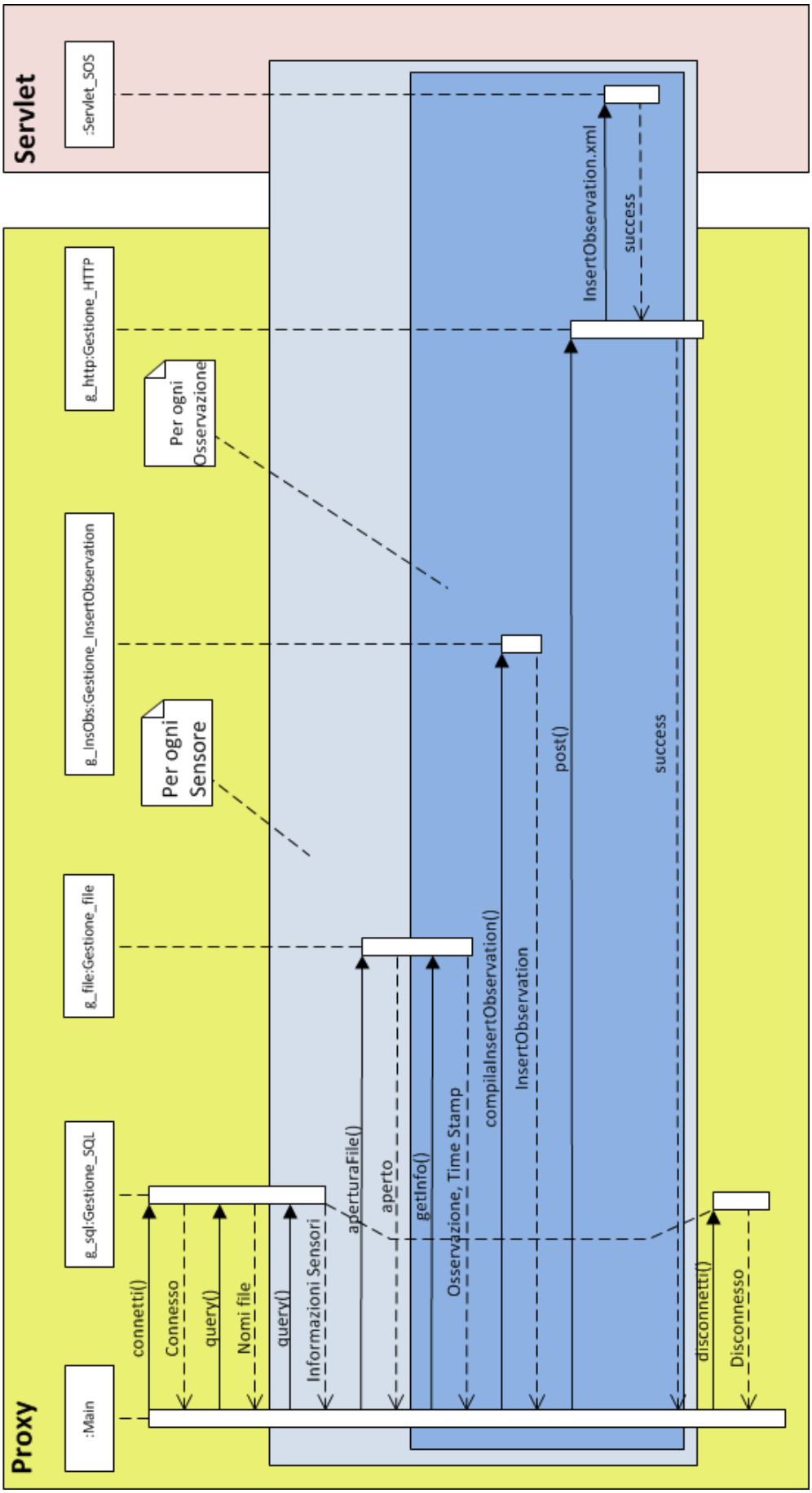

Figura 33: Diagramma di sequenza dell'inserimento di un'osservazione nel database SOS.

6.4 SITO WEB

In questa sezione è analizzato con maggior dettaglio l'implementazione del **Sito Web**. Il codice è stato sviluppato da un terzo mentre le funzionalità e alcuni algoritmi sono stati progettati all'interno della presente tesi.

6.4.1 *Home Page*

Nella home page del **Sito Web** è offerta la barra di navigazione del sito (lato sinistro) e le informazioni sulle stazioni meteorologiche considerate (al centro), come illustrato nella Figura 34.

Figura 34: Screenshot Homepage del **Sito Web**.

Nella barra di navigazione sono elencate le operazioni che può compiere l'utente:

- Stazioni meteorologiche, accedere alle informazioni di base riguardanti le stazioni meteorologiche gestite dal **Sito Web**
- Registrazione Sensore, permette di registrare un nuovo sensore ad una stazione meteorologica
- Grafici, visualizza i grafici con i valori degli ultimi tre giorni delle osservazioni prese dalle stazioni meteorologiche
- Visualizzazione osservazioni, ricerca e visualizzazione di osservazioni impostando dei filtri
- Storico, permette di scaricare i file compressi delle osservazioni acquisite dalle stazioni Meteorologiche.

6.4.2 *Registrazione Sensore*

Per la compilazione della richiesta di *RegisterSensor*, l'idea è la stessa adoperata dal **Proxy** per la *InsertObservation*, ovvero:
è adoperato un *template* [vedi Listing 16] dove i campi variabili sono rappresentati da elementi prefissati che, al momento della richiesta, sono sostituiti con i parametri inseriti nella form dall'utente.
Un elenco dei tag speciali sostituiti nel template [vedi sottosottosezione 9.1.2.1] sono visualizzati nella Tabella 14.

Tabella 14: Tabella che contiene gli elementi che saranno sostituiti nel *template* della *RegisterSensor*.

Campo	**Significato**
<IDENTIFIER>	ID del Sensore
<STATUS>	Stato del Sensore (attivo/disattivo)
<MOBILE>	Mobilità del Sensore
<DESCRIZIONE-FOI>	Breve descrizione della AWS
<FOI>	Identificativo della AWS
<NOME>	Nome Sensore
<LAT>	Latitudine
<LON>	Longitudine
<ALT>	Altitudine
<SEZIONE_INPUT>	Elenco dei fenomeni in ingresso al sensore
<SEZIONE_OUTPUT>	Elenco dei fenomeni in uscita dal sensore

É importante sottolineare che questo processo genera automaticamente anche un file SensorML di descrizione del sensore. Questo file è salvato in un'opportuna cartella del SOS e viene restituito in seguito ad una *DescribeSensor* dove è stato specificato l'identificativo del sensore come valore dell'elemento *<swes:procedure></swes:procedure>*.
Contestualmente alla registrazione è eseguita anche una operazione di *insert* nel database interno *appoggio_SOS*.
La richiesta di *RegisterSensor* alla servlet è implementata tramite socket [4].

4 Questo vale anche per le altre richieste inoltrate alla servlet.

6.4.3 *Acquisizione delle Capabilities e compilazione dei Filtri*

Per acquisire le informazioni necessarie ad inizializzare i *radio* della pagina HTML che offre i filtri per la ricerca delle osservazioni, è necessario eseguire una richiesta preventiva di *GetCapabilities* [vedi sottosezione 9.1.1] alla servlet che gestisce il SOS. Dalla risposta sono estrapolate le informazioni riguardanti:

- Stazione Meteorologica ⟶ rappresenta il *Feature Of Interest (FOI)* dove l'osservazione è stata campionata.
  - Selezione obbligatoria e singola
- Sensori ⟶ rappresentano le *procedure* che hanno campionato l'osservazione
  - Selezione facoltativa e multipla
- Fenomeni ⟶ rappresentano le *observedProperty* campionate dal sensore
  - Selezione facoltativa e multipla
- Vincolo temporale ⟶ rappresenta il *time* in cui l'osservazione è stata campionata
  - Selezione facoltativa
  - Se è impostato solo il valore massimo ⇒ sono visualizzate tutte le osservazioni campionate ad istanti successivi
  - Se è impostato solo il valore minimo ⇒ sono visualizzate tutte le osservazioni campionate ad istanti precedenti
  - Se sono impostati entrambi i valori ⇒ sono visualizzate tutte le osservazioni campionate negli istanti compresi nell'intervallo

6.4.4 *Visualizzazione Osservazioni*

Previa la corretta selezione dei filtri, è inoltrata alla servlet una richiesta di *GetObservation*. La compilazione della richiesta avviene in automatico, così come per la *RegisterSensor* [vedi sottosezione 6.4.2]. Un elenco dei tag speciali sostituiti nel template [vedi sottosottosezione 9.1.4.1] sono visualizzati nella Tabella 15.

Tabella 15: Tabella che contiene gli elementi che saranno sostituiti nel *template* della *GetObservation*.

Campo	Significato	Cardinalità
<FEATURE_OF_INTEREST>	AWS	1:1
<PROCEDURE>	Sensore	0:N
<OBSERVED_PROPERTY>	Fenomeni	0:N
<TEMPORAL_FILTER>	Filtro Temporale	0:2

6.4.5 *Grafici*

I grafici sono ottenuti con il *rrdtool*, che è un'estensione dinamica del PHP. É stata scelta questa libreria perché permette una gestione più semplice dell'aggiornamento automatico dei grafici al trascorrere del tempo. In particolare, con cadenza giornaliera:
è eseguito uno script che si occupa di recuperare i dati dal database del SOS e richiamare l'*update* dei grafici.

6.5 CR1000

Il *CR1000* è l'elemento centrale delle stazioni meteorologiche prese in considerazione nella presente tesi. Su tali datalogger è in esecuzione un programma che si occupa del sensing e dell'immagazzinamento delle osservazioni acquisite dai sensori connessi.
Purtroppo, durante la stesura del codice non è stato a disposizione nè un *CR1000* nè un suo emulatore, bensì solo un compilatore. Quindi del seguente codice è garantita solo la correttezza sintattica ma non semantica.
Nei successivi paragrafi sono evidenziati le modifiche apportate al codice attualmente in esecuzione sulla stazione meteorologica in Val de La Mare.

6.5.1 *Segmentazione del Codice*

Per ragioni di leggibilitè, il codice, è stato diviso su più file contenenti:

- Le costanti, le variabili e le corrispondenti unità di misura
- Gli statement per il sensing dell'ambiente
- Gli statement per l'immagazzinamento dei dati
- Gli statement per la gestione della comunicazione

La suddivisione su più file è resa possibile grazie all'istruzione *Include*:

Listing 9: Istruzione include del CRBASIC

```
' Sintassi:
'       Include "Device:Filename"
' Esempio:
Include "CPU:IF_Costanti&Variabili.CR1"
```

6.5.2 *Elisione di variabili e di istruzioni inutili*

L'analisi del codice attualmente in esecuzione sul *CR1000*, ha evidenziato l'inutilità di alcune variabili, che non sono nè lette nè immagazzinate, ed istruzioni di calcolo, i cui risultati non sono usati. Quindi è stata fatta una pulizia del codice che teoricamente permette di velocizzare l'esecuzione del programma e ridurre il *duty cycle* del *CR1000*.

6.5.3 *Perfezionamento del software della AWS*

Il funzionamento di un semplice programma CRBASIC è accennato nella sezione 3.3 e comporta:

- Campionamento
- Immagazzinamento applicando una funzione di aggregazione

Queste due fasi sono indipendenti l'una dall'altra. Esiste solo un vincolo temporale per cui il rate di immagazzinamento dev'essere multiplo di quello di campionamento. Il programma eseguito dalla Automatic Weather Station (AWS) preleva letture dai sensori ogni minuto e le immagazzina ogni cinque.
Il nivometro ha un rate di immagazzinamento orario e applica come funzione di aggregazione la *Sample*, che si limita a salvare solo l'ultimo valore letto dal sensore rendendo vane le precedenti letture. É per questo motivo che il suo rate di campionamento[5] è stato abbassato ad *una volta ogni ora*. Tenendo conto che il nivometro è il sensore che consuma di più, la modifica comporta un risparmio energetico senza intaccare la qualità e la quantità dei dati immagazzinati.
Il codice esegue il sensing della temperatura all'interno del *wiring panel*. Tali valori non sono usati (nè letti nè immagazzinati). Anziché eliminare la variabile che la contiene, è stata aggiunta nella tabella delle osservazioni campionate dai termistori.

5 frequenza alla quale un sensore acquisisce misure dall'ambiente.

6.5.4 *Introduzione della comunicazione*

La modifica più rilevante apportata al codice è stata la sezione che permette al *CR1000* di comunicare.
Il *CR1000*, è programmato per trasmettere a mezzogiorno in modo da sfruttare lo *zenit* e quindi il picco di energia fornito dal pannello solare.
Il CRBASIC supporta nativamente i modem satellitari che fanno riferimento alle tecnologie:

- Geostationary Operational Environmental Satellite (GOES)
- INMARSAT-C [dove INternational MARitime Satellite organization (INMARSAT)]
- Advanced Research and Global Observation Satellite (ARGOS)
- OmniSat

Il modem Iridium non è nativamente supportato, quindi è necessario esplicitare una comunicazione seriale [Interfaccia RS232] ed usare comandi AT.
Per ragioni di modularità e di sicurezza del codice, la sezione del programma che implementa la comunicazione è eseguita in *slow sequence*. Questa modalità ha priorità inferiore rispetto a quella del corpo del programma, data dallo statement *Scan()*.
La comunicazione è strutturata in tre *Subroutine*:

- Connessione
- Invio_Dati
- Disconnessione

Prima di analizzare la subroutine è utile analizzare il codice principale:

Listing 10: Codice CRBASIC principale che implementa la comunicazione

```
SlowSequence
  Dim isConnected
  Dim RN_1 As LONG
  Dim RN_2 As LONG
  Dim RN_3 As LONG
  Dim RN_4 As LONG
  Dim RN_5 As LONG
  Dim RN_6 As LONG
  Dim RN_7 As LONG

  ' Tempo massimo di connessione del modem
  Timeout = 10
```

```
  ' Lo Scan definisce la risoluzione temporale
  Scan(144, Min, 3, 0)
        ' Intervallo di accensione del modem
        If TimeIntoInterval(0, Rate_Comunicazione, Min) Then
          ' Accensione porta 1
          PortsConfig(&B1, 1)
          PortSet(1, true)

          ' Apertura connessione seriale e impostazione del
          ' baud rate. La dimensione del buffer è 2000
          SerialOpen(Porta_Modem, Modem_Baud, 0, 0, 2000)

          ' Dà al modem 20 secondi per accendersi
          Delay(1, 20, sec)

          SerialFlush(Porta_Modem)

          ' Connessione con il remote user
          Call Connessione(isConnected)

          If isConnected = 1 Then
            ' Invio dati
            Call Invio_Dati(RN_1, RN_2, RN_3, RN_4, RN_5,
                  RN_6, RN_7)

            ' Disconnessione del modem
            Call Disconnessione
          EndIf

          ' Operazioni di chiusura
          PortSet(1, False)
          SerialClose(Porta_Modem)
        End If

        ' Timeout di sicurezza per spegnere il modem
        If TimeIntoInterval(Timeout,
          Rate_Comunicazione(Mode), Min) Then

          Call Disconnessione
          ' Operazioni di chiusura
          PortSet(1, False)
          SerialClose(Porta_Modem)
        End If

  NextScan

EndProg
```

L'algoritmo che emerge dal Listing 10 può essere chiarito con il diagramma di flusso in Figura 35. Il modem è collegato al wiring panel del *CR1000* alla porta uno. L'alimentazione allo stesso è fornita solo quando deve comunicare. In questo modo è ottenuto un consi-

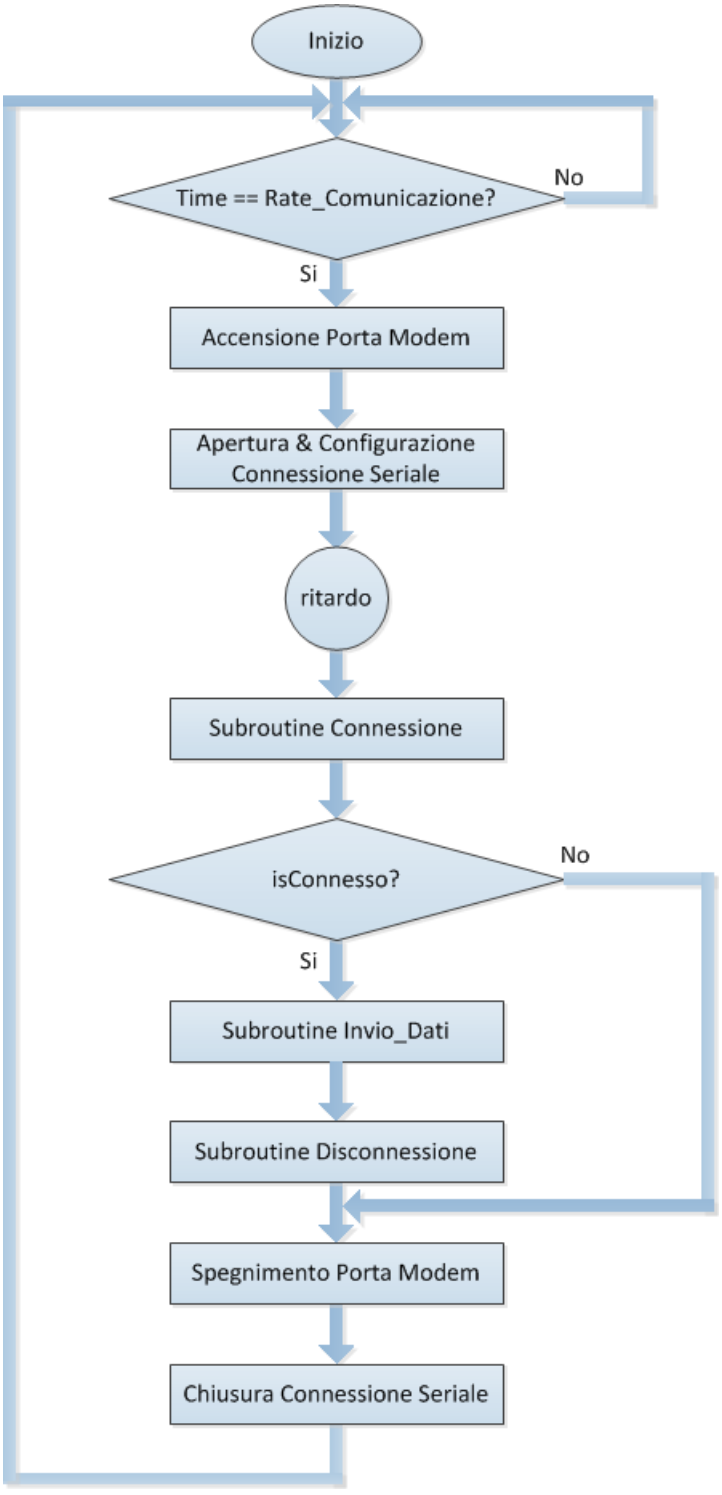

Figura 35: Diagramma di flusso della comunicazione implementata in CRBASIC.

derevole risparmio energetico eliminando un consumo giornaliero di 14.93mA perché il modem non si trova mai nello stato *sleep*. Questo risparmio è degno di nota in quanto rappresenta più della metà del consumo medio giornaliero del sistema.
I parametri configurabile della connessione seriale sono il *baud rate* e la capacità del buffer, espressa in byte.

6.5.4.1 *Subroutine Connessione*

Questa subroutine [Listing 11] stabilisce una connessione con la rete satellitare. Il comando AT da utilizzare è *ATDT* seguito dal numero telefonico che identifica il modem remoto. Perché la connessione vada a buon fine dev'essere ricevuta la stringa *CONNECT* entro due minuti.
La procedura di connessione è tentata per *Num_Max_Tentativi*, costante configurabile.

Listing 11: Codice CRBASIC della subroutine Connessione

```
Sub Connessione(tmp_isConnected)
        Dim tmp_isConnected = 1
        Dim tmp_num_tentativi
        Dim tmp_cont
        Dim tmp_AT_cmd As String * 40
        Dim str As String * 100

        tmp_AT_cmd = "ATDT " + Remote_Phone_Number + Chr(13) +
            Chr(10)

        ' Ciclo che tenta la connessione
        For tmp_num_tentativi = 1 to Num_Max_Tentativi step 1
                SerialFlush(Porta_Modem)
                SerialOut(Porta_Modem, tmp_AT_cmd, "", 0, 100)
                SerialIn(str, Porta_Modem, 1200, "CONNECT", 100)

                For tmp_cont = 1 to 90 step 1
                        If mid(str, tmp_cont, 7) = "CONNECT" Then
                                ExitSub
                        EndIf
                Next tmp_cont
        Next tmp_num_tentativi

        ' Se il controllo ha esito positivo => la connessione è
            fallita
        If tmp_num_tentativi > Num_Max_Tentativi Then
                tmp_isConnected = 0
        End If
Exit Sub
End Sub
```

6.5.4.2 *Subroutine Invio_Dati*

Questa subroutine invia i dati immagazzinati nelle tabelle. I dati sono inviati un record alla volta. Quest'ultimi sono prelevati dalle tabelle grazie all'istruzione *GetRecord()*, che prende come parametri:

- La variabile in cui salvare il record
- Il nome della tabella da cui prelevare la riga
- Il RN da cui iniziare a prelevare i record

Visto che la connessione con la rete satellitare può fallire, è stato previsto un meccanismo per essere certi di trasmettere tutte le osservazioni immagazzinate.
Per farlo è mantenuta traccia dell'ultimo RN spedito per ogni tabella e il prelevamento dei record ricomincia da tale indice. Per non rischiare di esaurire la batteria, nel caso in cui la stazione meteorologica rimanga isolata per diversi giorni e accumoli molti dati, è impostato un limite massimo di record da mandare giornalmente. Il limite è fissato a 4500, che indica la trasmissione di al più 90kB che necessitano un tempo di trasmissione di cinque minuti. L'incertezza è dovuta al fatto che i record non hanno tutti la stessa dimensione. Il calcolo è fatto nel *worst case*, ovvero considerando un record di dimensione 20B [tabella dei termoigrometri].

Listing 12: Codice CRBASIC della subroutine Invio_Dati

```
Sub Invio_Dati(tmp_RN_1, tmp_RN_2, tmp_RN_3, tmp_RN_4, tmp_RN_5,
    tmp_RN_6, tmp_RN_7)
        Dim tmp_RN_1
        Dim tmp_RN_2
        Dim tmp_RN_3
        Dim tmp_RN_4
        Dim tmp_RN_5
        Dim tmp_RN_6
        Dim tmp_RN_7

        Dim tmp_cont
        Dim tmp_ultimo_RN
        Dim tmp_record As String * 1000

        ' Invio dati tabella Nivometro
        tmp_ultimo_RN = Tab_Nivometro.Record
        Do While tmp_RN_1 <= tmp_ultimo_RN AND tmp_cont <=
            Num_Max_Record
                ' Recupero i dati dalla tabella e...
                GetRecord(tmp_record, Tab_Nivometro, tmp_RN_1)
                ' ...l'invio
                SerialOut(Porta_Modem, tmp_record, "", 0, 20)
                tmp_RN_1 = tmp_RN_1 + 1
                tmp_cont = tmp_cont + 1
```

```
        Loop

        ' Invio dati tabella Albedometro
        tmp_ultimo_RN = Tab_Albedometro.Record
        Do While tmp_RN_2 <= tmp_ultimo_RN AND tmp_cont <=
            Num_Max_Record
                ' Recupero i dati dalla tabella e...
                GetRecord(tmp_record, Tab_Albedometro, tmp_RN_2)
                ' ...l'invio
                SerialOut(Porta_Modem, tmp_record, "", 0, 12)
                tmp_RN_2 = tmp_RN_2 + 1
                tmp_cont = tmp_cont + 1
        Loop

        ' Invio dati tabella Termoigrometri
        tmp_ultimo_RN = Tab_Termoigrometri.Record
        Do While tmp_RN_3 <= tmp_ultimo_RN AND tmp_cont <=
            Num_Max_Record
                ' Recupero i dati dalla tabella e...
                GetRecord(tmp_record, Tab_Termoigrometri,
                    tmp_RN_3)
                ' ...l'invio
                SerialOut(Porta_Modem, tmp_record, "", 0, 20)
                tmp_RN_3 = tmp_RN_3 + 1
                tmp_cont = tmp_cont + 1
        Loop

        ' Invio dati tabella Anemometro
        tmp_ultimo_RN = Tab_Anemometro.Record
        Do While tmp_RN_4 <= tmp_ultimo_RN AND tmp_cont <=
            Num_Max_Record
                ' Recupero i dati dalla tabella e...
                GetRecord(tmp_record, Tab_Anemometro, tmp_RN_4)
                ' ...l'invio
                SerialOut(Porta_Modem, tmp_record, "", 0, 16)
                tmp_RN_4 = tmp_RN_4 + 1
                tmp_cont = tmp_cont + 1
        Loop

        ' Invio dati tabella Pirgeometro
        tmp_ultimo_RN = Tab_Pirgeometri.Record
        Do While tmp_RN_5 <= tmp_ultimo_RN AND tmp_cont <=
            Num_Max_Record
                ' Recupero i dati dalla tabella e...
                GetRecord(tmp_record, Tab_Pirgeometri, tmp_RN_5)
                ' ...l'invio
                SerialOut(Porta_Modem, tmp_record, "", 0, 16)
                tmp_RN_5 = tmp_RN_5 + 1
                tmp_cont = tmp_cont + 1
        Loop

        ' Invio dati tabella Termistori
```

```
        tmp_ultimo_RN = Tab_Termistori.Record
        Do While tmp_RN_6 <= tmp_ultimo_RN AND tmp_cont <=
            Num_Max_Record
                ' Recupero i dati dalla tabella e...
                GetRecord(tmp_record, Tab_Termistori, tmp_RN_6)
                ' ...l'invio
                SerialOut(Porta_Modem, tmp_record, "", 0, 14)
                tmp_RN_6 = tmp_RN_6 + 1
                tmp_cont = tmp_cont + 1
        Loop

        ' Invio dati tabella Batteria
        tmp_ultimo_RN = Tab_Batteria.Record
        Do While tmp_RN_7 <= tmp_ultimo_RN AND tmp_cont <=
            Num_Max_Record
                ' Recupero i dati dalla tabella e...
                GetRecord(tmp_record, Tab_Batteria, tmp_RN_7)
                ' ...l'invio
                SerialOut(Porta_Modem, tmp_record, "", 0, 12)
                tmp_RN_7 = tmp_RN_7 + 1
                tmp_cont = tmp_cont + 1
        Loop

Exit Sub
End Sub
```

6.5.4.3 *Subroutine Disconnessione*

Questa subroutine effettua la disconnessione dalla rete satellitare. I comandi AT da usare sono due:

- +++, per uscire dalla modalità data ed entrare in quella di inserimento di comandi
- ATH, per la disconnessione dalla rete satellitare

A livello di codice:

Listing 13: Codice CRBASIC della subroutine Disconnessione

```
Sub Disconnessione
        Dim tmp_str As String * 40
        Dim tmp_AT_cmd As String * 40
        tmp_AT_cmd = "+++"

        ' Usciamo dalla modalità data
        SerialOut(Porta_Modem, tmp_AT_cmd, "", 0, 100)
        SerialFlush(Porta_Modem)
        ' Attendo (10 secondi) la conferma
        SerialIn(tmp_str, Porta_Modem, 1000, "OK", 100)

        tmp_AT_cmd = "ATH" + Chr(13) + Chr(10)
```

```
        ' Disconnessione
        SerialOut(Porta_Modem, tmp_AT_cmd, "", 0, 100)
        SerialFlush(Porta_Modem)
        ' Attendo (10 secondi) la conferma
        SerialIn(tmp_str, Porta_Modem, 1000, "OK", 100)
Exit Sub
End Sub
```

6.5.4.4 *Meccanismo di interruzione della comunicazione*

Per evitare situazioni erronee che esauriscano la batteria è stato implementato un meccanismo per interrompere la comunicazione dopo un lasso prefissato di tempo. A livello di codice:

Listing 14: Codice che spegne il modem dopo un lasso di tempo prefissato che la comunicazione è cominciata

```
' Timeout di sicurezza per spegnere il modem
If TimeIntoInterval(Timeout, Rate_Comunicazione(Mode), Min) Then
        Call Disconnessione

        ' Operazioni di chiusura
        PortSet(1, False)
        SerialClose(Porta_Modem)
End If
```

6.5.5 *Modalità a risparmio energetico*

É stato implementato un meccanismo adattivo che, a seconda del valore di tensione della batteria, fa passare la stazione meteorologica ad una modalità di funzionamento a risparmio energetico. L'idea è semplice:
il sensing della batteria avviene una volta al minuto, se il valore misurato scende al di sotto di una soglia prefissata allora si attiva la modalità a risparmio energetico; mentre, quando la situazione si ristabilizza, è ripristinata la consueta modalità di funzionamento.
Nella modalità a risparmio energetico sono considerevolmente abbassati i rate di campionamento e di immagazzinamento, così come la frequenza con cui avviene la comunicazione.

6.5.6 *Riprogrammazione da remoto*

Il codice descritto è stato implementato di modo che non siano necessarie troppe modifiche per integrare un'eventuale riprogrammabilità da remoto dei sensori annessi al *CR1000*. Questa è la motivazione principale alla suddivisione in tabelle per tipologia di sensori. Così i rate di campionamento e di immagazzinamento dei sensori sono del tutto dissociati tra loro.

7 TEST

In questo capitolo è sottolineata l'importanza della fase di test di un'applicazione e i problemi di indisponibilità dell'hardware riscontrati.

7.1 PROBLEMA: INDISPONIBILITÀ HARDWARE

Prima di acquistare un hardware costoso è sempre necessaria una fase di esplorazione e di approfondimento, per capirne le potenzialità e se l'acquisto è giustificato. Durante la tesi, è stato appreso come il sensing in ambito ambientale sia un terreno fertile per la ricerca e che il *CR1000* è un prodotto versatile e utile per proseguire su questa strada. Questa conclusione è giunta dopo un lungo lavoro di documentazione e i tempi di consegna dell'hardware sono lunghi. Quindi tutto il codice che è stato sviluppato non è stato testato. Addirittura, come accennato nella sezione 6.5, del codice CRBASIC è stato possibile solo un controllo sintattico. Invece, per quanto riguarda il **Proxy** e il **Sito Web**, è stato possibile effettuare dei test, parziali, in locale.
Il test è una fase indispensabile prima di procedere ad un'installazione su campo, soprattutto quando i siti sono difficilmente raggiungibili come in questo caso. Nelle applicazioni di sensing va considerato il rischio aggiuntivo di perdere dati o di corromperli, procurando un danno per la comunità scientifica a cui sono destinati.

7.2 INIZIALIZZAZIONE DEL DATABASE DEL SOS

Per testare correttamente il servizio è stato realizzato uno script MySQL per riempire il database del SOS con i sensori presenti sulla stazione meteorologica in Val de La Mare. Sempre tramite script, sono state aggiunte delle osservazioni ad ogni sensore. In questo modo è stato creato uno scenario realistico e riproducibile di come il database del SOS si troverà ad operare in una situazione di regime.
Da sottolineare che non è stata trovata una definizione per tutti i fenomeni misurati dalla Automatic Weather Station (AWS) nel dizionario on-line EDCS. In particolare, i fenomeni che sono stati inseriti senza specificare un Uniform Resource Name (URN), sono:

- *WIND_DIRECTION_STD_DEV*, deviazione standard della direzione del vento
- *RADIATIVE_FLUX_DOWNWELLING_IN*, radiazione a onde corte in ingresso

- *RADIATIVE_FLUX_DOWNWELLING_OUT*, radiazione a onde corte in uscita
- *RADIATIVE_FLUX_UPWELLING_IN*, radiazione a onde lunghe in ingresso
- *RADIATIVE_FLUX_UPWELLING_OUT*, radiazione a onde lunghe in uscita

7.3 TEST DEL PROXY

Tutte le funzionalità offerte dal **Proxy** sono state testate in locale. L'unica parte mancante è la comunicazione con il *CR1000* che, per come è strutturato il codice, è l'evento che guida tutte le altre operazioni. Questa è stata simulata popolando con dei file prescritti la cartella in cui il **Proxy** immagazzinaerà le osservazioni ricevute. Lo scorrere del tempo è stato simulato con un ciclo *while*. Ad ogni iterazione, teoricamente, corrisponde un giorno.
Il test ha dimostrato la correttezza:

- Dell'inserimento delle osservazioni all'interno del database dei SOS
- Della gestione di eventuali ritrasmissioni
- Dell'archiviazione dei dati nello storico
- Della compilazione dei file di Log
- Del servizio di notifica via posta elettronica

7.4 TEST DEL SITO WEB

Tutti i servizi offerti dal **Sito Web** sono state testati in locale. In particolare è stata verificata la correttezza delle procedure di:

- Registrazione di un sensore al servizio SOS
- Compilazione della descrizione SensorML del sensore
- Visualizzazione dei grafici
- Ricerca e visualizzazione tabellare delle osservazioni impostando dei filtri
- Download dei file archiviati nello storico

Parte III

CONCLUSIONI

In questa parte sono presentate le conclusioni e le possibili estensioni del servizio progettato.

8 CONCLUSIONI E SVILUPPI FUTURI

8.1 CONCLUSIONI

L'obiettivo cardine, a cui ruota attorno la tesi, è l'evoluzione di una stazione meteorologica, basata su datalogger, in una capace di comunicare con l'esterno appoggiandosi su un servizio di middleware che permette la raccolta e l'inserimento delle osservazioni in un database standard quale il SOS. Lo scopo è dunque ambizioso per essere realizzato completamente in un'unica tesi. Questo lavoro, quindi, va considerato come un progetto pilota nell'ambito del sensing ambientale e come l'inizio di una collaborazione multidisciplinare con i colleghi geologi.
Detto questo, nella presente tesi, è stato progettato un servizio per la gestione di una Automatic Weather Station (AWS) che invia, grazie alla comunicazione satellitare, osservazioni ambientali immagazzinandole in un database standard e un sito web che ne permette la visualizzazione. Lato implementativo il servizio è stato realizzato completamente eccetto la parte di comunicazione legata al **Proxy**, in quanto la scelta del modem non è ancora del tutto definita.
É stata acquisita molta esperienza, utile nel continuare la via tracciata da questo prototipo; in particolare:

- la fattibilità e la flessibilità della comunicazione satellitare in ambienti ostici per la raggiungibilità da parte sia dell'uomo che delle infrastrutture elettriche
- l'importanza di usare standard per la descrizione dei sensori e delle osservazioni che acquisiscono, di modo da rendere il servizio universalmente valido e comprensibile
- la fertilità intellettuale di incontri multidisciplinare da cui scaturiscono idee innovative

8.2 PROPOSTE PER SVILUPPI FUTURI

Vista l'indisponibilità dell'hardware [vedi sezione 7.1] del *CR1000* e del modem satellitare, i primi step da realizzare sono il test del software prodotto e l'implementazione, nel **Proxy**, della comunicazione con la stazione meteorologica.
Due sono le proposte per ampliare il **Proxy**.
Rendere la AWS riprogrammabile da remoto per rendere il servizio completo e incrementarne l'utilità. Infatti, sarebbe possibile configurare parametri quali la frequenza di campionamento e di immagaz-

zinamento dei sensori, senza recarsi in loco. Lo stesso dicasi per la frequenza con cui il *CR1000* comunica.
Implementare un algoritmo di compressione dei dati per ridurre non solo i consumi ma soprattutto i costi dovuti alla comunicazione satellitare.
Mentre, per quanto riguarda il **Sito Web**, un'idea potrebbe essere arricchire l'interfaccia di registrazione del sensore. Ovvero riempire i menù a tendina dei fenomeni e delle unità di misura corrispondenti, prendendo i valori direttamente dal dizionario dell'EDCS.

Parte IV

APPENDICE

9 APPENDICE

9.1 FILE XML PER L'INTERAZIONE CON IL SOS

In questa sezione sono riportati i file XML delle richieste effettuate alla servlet che implementa il servizio SOS e gli eventuali template adoperati per la loro compilazione.

9.1.1 *GetCapabilities*

La *GetCapabilities* è una delle richieste predefinite per la servlet che implementa il servizio SOS. Questa operazione permette di recuperare le informazioni relative al servizio SOS, tra cui:

- Stazioni Meteorologiche
- Sensori
- Operazioni Permesse
- ...

Fa parte del profilo *core*. Ha la seguente struttura:

Listing 15: Richiesta di GetCapabilities

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
  xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2003/05/
    soap-envelope
    http://www.w3.org/2003/05/soap-envelope"
  xmlns:sos="http://www.opengis.net/sos/1.0"
  xmlns:ows="http://www.opengis.net/ows/1.1">

  <env:Body>
        <GetCapabilities
          xmlns="http://www.opengis.net/sos/2.0"
          xmlns:ows="http://www.opengis.net/ows/1.1"
          xmlns:ogc="http://www.opengis.net/ogc"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.opengis.net/sos/2.0
        http://schemas.opengis.net/sos/2.0.0/
                    sosGetCapabilities.xsd"
                service="SOS">
```

```
          <ows:AcceptVersions>
                <ows:Version>2.0.0</ows:Version>
          </ows:AcceptVersions>

          <ows:Sections>
                <ows:Section>OperationsMetadata</ows:Section>
                <ows:Section>ServiceIdentification</ows:Section>
                <ows:Section>ServiceProvider</ows:Section>
                <ows:Section>FilterCapabilities</ows:Section>
                <ows:Section>Contents</ows:Section>
          </ows:Sections>

        </GetCapabilities>
  </env:Body>
</env:Envelope>
```

9.1.2 *RegisterSensor*

La *RegisterSensor* è una delle richieste predefinite per la servlet che implementa il servizio SOS. Questa operazione permette di registrare un nuovo sensore al servizio SOS e fa parte del profilo *transactional*. Durante la fase di registrazione dev'essere specificato un identificatore. Questo dev'essere riportato in tutte le richieste che riguardano il sensore, quindi sia per recuperare le sue *capabilities* che per inserire osservazioni.

9.1.2.1 *Template RegisterSensor*

Template della richiesta *RegisterSensor*. La manipolazione del file avviene in seguito alla procedura di registrazione di un sensore, operazione offerta dal **Sito Web**. L'utente riempie una form e i parametri sono mappati con i tag speciali [vedi Tabella 14] del template sottostante.

Listing 16: Template della richiesta di RegisterSensor

```
<RegisterSensor xmlns="http://www.opengis.net/sos/1.0"
xmlns:gml="http://www.opengis.net/gml"
xmlns:ogc="http://www.opengis.net/ogc"
xmlns:om="http://www.opengis.net/om/1.0"
xmlns:ows="http://www.opengeospatial.net/ows"
xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
xmlns:swe="http://www.opengis.net/swe/1.0.1"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  service="SOS" version="1.0.0"
xsi:schemaLocation="http://www.opengis.net/sos/1.0
  http://schemas.opengis.net/sos/1.0.0/
    sosRegisterSensor.xsd
  http://www.opengis.net/om/1.0
  http://schemas.opengis.net/om/1.0.0/extensions/
```

```
  observationSpecialization_override.xsd">

<SensorDescription>
      <sml:SensorML version="1.0.1">
        <sml:member>
              <sml:System>
                <sml:identification>
                      <sml:IdentifierList>
                        <sml:identifier>
                              <sml:Term
                                definition="urn:ogc:def:
                                    identifier:
                                  OGC:uniqueID">
                                <sml:value>urn:ogc:object:
                                    feature:
                                  Sensor:Ing-Pi:<IDENTIFIER>
                                </sml:value>
                              </sml:Term>
                        </sml:identifier>
                      </sml:IdentifierList>
                </sml:identification>
                <sml:capabilities>
                  <swe:SimpleDataRecord>
                        <swe:field name="status">
                              <swe:Boolean>
                                <swe:value><STATUS></swe:value>
                              </swe:Boolean>
                        </swe:field>
                        <swe:field name="mobile">
                              <swe:Boolean>
                                <swe:value><MOBILE></swe:value>
                              </swe:Boolean>
                        </swe:field>
                      </swe:SimpleDataRecord>
                      <swe:SimpleDataRecord>
                        <swe:field name="FOI_ID">
                          <swe:Text definition="<DESCRIZIONE_
                              FOI>">
                              <swe:value><FOI></swe:value>
                            </swe:Text>
                        </swe:field>
                  </swe:SimpleDataRecord>
                </sml:capabilities>
                <sml:position name="<NOME>_Position">
                      <swe:Position
                        referenceFrame="urn:ogc:def:crs:EPSG
                            ::4326">
                        <swe:location>
                              <swe:Vector gml:id="
                                  STATION_LOCATION">
                                <swe:coordinate
                                  name="longitude">
```

```
                                        <swe:Quantity axisID="x">
                                          <swe:uom code="degree"
                                              />
                                          <swe:value>
                                            <LAT>
                                          </swe:value>
                                        </swe:Quantity>
                                    </swe:coordinate>
                                    <swe:coordinate
                                      name="latitude">
                                        <swe:Quantity axisID="y">
                                          <swe:uom code="degree"
                                              />
                                          <swe:value>
                                            <LON>
                                          </swe:value>
                                        </swe:Quantity>
                                    </swe:coordinate>
                                    <swe:coordinate
                                      name="altitude">
                                        <swe:Quantity axisID="z">
                                          <swe:uom code="m"/>
                                          <swe:value>
                                            <ALT>
                                          </swe:value>
                                        </swe:Quantity>
                                    </swe:coordinate>
                                </swe:Vector>
                            </swe:location>
                        </swe:Position>
                    </sml:position>
                    <sml:inputs>
                        <sml:InputList>
                          <SEZIONE_INPUT>
                        </sml:InputList>
                    </sml:inputs>
                    <sml:outputs>
                        <sml:OutputList>
                          <SEZIONE_OUTPUT>
                        </sml:OutputList>
                    </sml:outputs>
                </sml:System>
            </sml:member>
        </sml:SensorML>
    </SensorDescription>
    <ObservationTemplate>
        <om:Measurement>
            <om:samplingTime/>
            <om:procedure/>
            <om:observedProperty/>
            <om:featureOfInterest/>
            <om:result uom="">0.0</om:result>
```

```
        </om:Measurement>
  </ObservationTemplate>
</RegisterSensor>
```

9.1.3 *InsertObservation*

La *InsertObservation* è una delle richieste predefinite per la servlet che implementa il servizio SOS. Questa operazione permette di inserire le osservazioni acquisite dai sensori all'interno del database del SOS e fa parte del profilo *transactional*.

9.1.3.1 *Template InsertObservation*

Template della richiesta *InsertObservation* . Viene manipolato dal **Proxy** per inserire correttamente le osservazioni nel database del SOS.

Listing 17: Template della richiesta di InsertObservation

```
<InsertObservation xmlns="http://www.opengis.net/sos/1.0"
  xmlns:ows="http://www.opengis.net/ows/1.1"
  xmlns:ogc="http://www.opengis.net/ogc"
  xmlns:om="http://www.opengis.net/om/1.0"
  xmlns:sos="http://www.opengis.net/sos/1.0"
  xmlns:sa="http://www.opengis.net/sampling/1.0"
  xmlns:gml="http://www.opengis.net/gml"
  xmlns:swe="http://www.opengis.net/swe/1.0.1"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.opengis.net/sos/1.0
  http://schemas.opengis.net/sos/1.0.0/sosInsert.xsd
  http://www.opengis.net/sampling/1.0
  http://schemas.opengis.net/sampling/1.0.0/sampling.xsd
  http://www.opengis.net/om/1.0
  http://schemas.opengis.net/om/1.0.0/extensions/
    observationSpecialization_override.xsd"
  service="SOS" version="1.0.0">

 <AssignedSensorId><ID_Sensore></AssignedSensorId>

 <om:Observation>
  <om:procedure xlink:href="<ID_Sensore>" />
  <om:observedProperty>
   <swe:CompositePhenomenon gml:id="fen-comp"
    dimension="1">
    <gml:name>resultComponents</gml:name>
        <Elenco_Fenomeni>
   </swe:CompositePhenomenon>
  </om:observedProperty>

  <om:featureOfInterest>
   <gml:FeatureCollection>
```

```
    <gml:featureMember>
     <sa:SamplingPoint gml:id="<FOI>">
      <gml:name><FOI></gml:name>
      <sa:sampledFeature xlink:href=""/>
      <sa:position>
       <gml:Point>
        <gml:pos srsName="urn:ogc:def:crs:EPSG::4326">
                  <lat> <lon>
                </gml:pos>
       </gml:Point>
      </sa:position>
     </sa:SamplingPoint>
    </gml:featureMember>
   </gml:FeatureCollection>
  </om:featureOfInterest>

  <om:result>
   <swe:DataArray>
    <swe:elementType name="Components">
     <swe:DataRecord>
      <swe:field name="Time">
       <swe:Time definition="http://www.opengis.net/
            def/uom/ISO-8601/0/Gregorian" />
      </swe:field>
      <swe:field name="feature">
       <swe:Text definition="http://www.opengis.net/
            def/property/OGC/0/FeatureOfInterest" />
      </swe:field>
      <Elenco_Campi>
     </swe:DataRecord>
    </swe:elementType>

    <swe:encoding>
     <swe:TextBlock decimalSeparator="."
           tokenSeparator="," blockSeparator=";" />
    </swe:encoding>

    <swe:values>
    <TS>,<FOI>,<Elenco_Valori>
    </swe:values>

   </swe:DataArray>
  </om:result>
 </om:Observation>
</InsertObservation>
```

9.1.4 *GetObservation*

La *GetObservation* è una delle richieste predefinite per la servlet che implementa il servizio SOS. Questa operazione permette di recuperare le osservazioni dal database del SOS e fa parte del profilo *core*. É

possibile configurare diversi filtri, tra cui:

- FOI, luogo in cui l'osservazione è stata campionata
- Procedure, sensore che ha campionato l'osservazione
- ObservedProperty, fenomeno a cui appartiene l'osservazione
- Offering, raggruppamento logico di osservazioni
- vincoli temporali, intervallo di tempo in cui l'osservazione viene campionata

9.1.4.1 *Template GetObservation*

Template della richiesta *GetObservation* . Viene manipolato dallo script PHP che gestisce il **Sito Web** per recuperare le osservazioni all'interno del SOS con dei filtri configurabili.

Listing 18: Template della richiesta di GetObservation

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
  xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2003/05/
    soap-envelope
    http://www.w3.org/2003/05/soap-envelope"
  xmlns:sos="http://www.opengis.net/sos/1.0"
  xmlns:ows="http://www.opengis.net/ows/1.1">
  <env:Body>
    <sos:GetObservation
          xmlns="http://www.opengis.net/sos/2.0"
          service="SOS" version="2.0.0"
          xmlns:sos="http://www.opengis.net/sos/2.0"
          xmlns:fes="http://www.opengis.net/fes/2.0"
          xmlns:gml="http://www.opengis.net/gml/3.2"
          xmlns:swe="http://www.opengis.net/swe/2.0"
          xmlns:swes="http://www.opengis.net/swes/2.0"
          xmlns:xlink="http://www.w3.org/1999/xlink"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.opengis.net/sos/2.0
                http://schemas.opengis.net/sos/2.0.0/sos.xsd">
          <!-- Fenomeni -->
          <OBSERVED_PROPERTY>

          <!--Sensori -->
          <PROCEDURE>

          <!-- Filtro temporale-->
          <TEMPORAL_FILTER>

          <!-- Stazione Meteorologica -->
```

```
          <FEATURE_OF_INTEREST>

          <responseFormat>
            http://www.opengis.net/om/2.0
          </responseFormat>
    </sos:GetObservation>
  </env:Body>
</env:Envelope>
```

Parte V

BIBLIOGRAFIA

BIBLIOGRAFIA

[1] Campbell Scientific. URL http://www.campbellsci.com/.

[2] EDCS. URL http://www.sedris.org/edcs.htm.

[3] ESA - Observing the Earth. URL http://www.esa.int/esaEO/index.html.

[4] Glacsweb. URL http://glacsweb.org/.

[5] Data from imja glacial lake. URL http://fsds.dc.affrc.go.jp/data4/Himalayan/.

[6] OGC. URL http://www.opengeospatial.org/.

[7] SIERRANET. URL http://systems.berkeley.edu/wsn/.

[8] WGMS. URL http://www.wgms.ch/.

[9] Doxygen. URL http://www.doxygen.org/.

[10] Eclipse - Webtool. URL http://www.eclipse.org/webtool.

[11] Graphviz. URL http://www.graphviz.org/.

[12] permaNET. URL http://www.permanet-alpinespace.eu/home.html.

[13] permasense. URL http://www.permasense.ch/.

[14] M. Botts and A. Robin. OpenGIS Sensor Model Language (SensorML) implementation specification. 2007. URL http://portal.opengeospatial.org/files/?artifact_id=21273.

[15] M. Botts, G. Percivall, C. Reed, and J. Davidson. OGC Sensor Web Enablement: Overview and high level architecture. 2007. URL http://portal.opengeospatial.org/files/?artifact_id=25562.

[16] Campbell Scientific. *CR1000 Measurement and Control System*, 2011. URL http://s.campbellsci.com/documents/us/manuals/cr1000.pdf.

[17] Campbell Scientific. *CS-GSM - Transceiver Kits*, 2011. URL ftp://ftp.campbellsci.com/pub/csl/outgoing/uk/manuals/cs_gsm.pdf.

[18] Campbell Scientific. Application power supply note. Technical report, 2011. URL http://s.campbellsci.com/documents/us/technical-papers/pow-sup.pdf.

[19] A. Hasler, I. Talzi, J. Beutul, C. Tschudin, and S. Gruber. Wireless Sensor Networks in permafrost reaserch - concept requirements, implementation and challanges. In *In: 9th International Conference on Permafrost*. URL http://www.tik.ee.ethz.ch/~beutel/pub/HTBTG2008.pdf.

[20] K. Martinez, R. Ong, and J. Hart. Glacsweb: A sensor network for hostile environments. *The First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, 2004. URL http://eprints.ecs.soton.ac.uk/9920/1/kirkmartinez.pdf.

[21] K. Martinez, P. Padhy, A. Riddoch, R. Ong, and J. Hart. Glacial environment monitoring using sensor networks. 2005. URL http://eprints.ecs.soton.ac.uk/10845/1/Glacsweb-_REALSWSN_Workshop_Paper.pdf.

[22] A. Na and M. Priest. Sensor Observation Service. 2007. URL http://portal.opengeospatial.org/files/?artifact_id=26667.

[23] M. Pun, B. Shrestha, G. R. Upadhaya, P. Manandhar, and I. Badal. Wireless networking and filed server in the high Himalayas. In *World conference on agricultural information and IT, IAALD AFITA WCCA 2008*. URL http://www.cabi.org/GARA/FullTextPDF/2008/20083298187.pdf.

[24] R.F. Richbourg, D. D. Miller, A. Janett, P. G. Foley, and D. Jodeit. The Environmental Data Coding Specification: The standard for specification of environmental objects and properties. 2002. URL http://www.genestrickland.com/geospatial/sedris/sedris_edcs.pdf.

[25] SNAMP Water Team. Sierra Nevada adaptive management project, 2010. URL http://snamp.cnr.berkeley.edu/static/documents/2010/11/29/Water_FieldTrip_Notes_11-18-2010.pdf.

[26] A. Whiteside. Definition identifier URNs in OGC namespace. 2007. URL http://portal.opengeospatial.org/files/?artifact_id=24045.

[27] Wireless Innovation. *MiChroSat 2403*, 2010. URL http://michrosat.com/downloads/MiChroSat2403Manual_Rev_1-2_021110.pdf.

[28] A Zappelli. *Installazione Sensor Observation Service*, 2011. URL https://vcs.ing.unipi.it/trac/environment/browser/svn-environment/zappelli/SWE/Produzione_propria/Installazione%20SOS.pdf.

Parte VI

RINGRAZIAMENTI

Considerate la vostra semenza: fatti non foste per vivere come bruti, ma per seguire virtute e canoscenza.

— Dante Alighieri

RINGRAZIAMENTI

Innanzitutto vorrei ringraziare la mia famiglia per l'appoggio e la fiducia dimostrata in tutti questi anni, sperando di non deluderla mai: tutti passano, Voi siete il mio unico punto fermo.
Vorrei ringraziare il professore Avvenuti che ha incanalato la mia curiosità relativa ai sensori in un ambito affascinante quale quello del monitoraggio ambientale, Daniel che mi ha seguito in questo progetto, sopportando una fitta comunicazione di e-mail :) e Vincenzo che si è occupato della parte web.
Per la sezione *svago* un grazie alle mie squadre con cui ho condiviso sfide ma soprattutto cene memorabili! e alle mie amiche, che mi hanno tenuta in giro in queste lunghe notti, sopportando il mio sarcasmo pungente, la mia superbia alla *Sheldon Cooper* e le battute più stupide che una mente umana (?) possa concepire!
Infine grazie Alessia per la tua presenza in tutti questi anni, l'unica che mi sopporta e non mi lascia indietro: A^2. Strano come le distanze fisiche siano davvero relative.
Di questi anni sarà impossibile dimenticare gli scalini del polo A [i più alti mai progettati!!], trenitalia con i suoi proverbiali *cinque minuti* di ritardo, citazioni celebri come "puntolini!!", "Interrupt disabilitate", "Architeccia"e la perfetta sincronia dei momenti yogurt a mensa divenuti un magico rendez-vous!
Concluderei con una citazione degna di un Ingegnere (anche se idealmente pronunciata da un fisico applicato):
"A volte dimentico che gli altri hanno dei limiti. É così triste..."